VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Předávání RTP toků v páteřní komunikaci na SIP

RTP proxying in SIP Backbone Communications

2019 Petr Vaněk

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Zadání bakalářské práce

Student: **Petr Vaněk**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2601R013 Telekomunikační technika

Téma: Předávání RTP toků v páteřní komunikaci na SIP
RTP Proxying in SIP Backbone Communications

Jazyk vypracování: čeština

Zásady pro vypracování:

Oblast VoIP technologií tvoří páteř moderních komunikací. Přenos hlasu po paketových sítích, navzdory počátečním problémům, představuje moderní, efektivní a ekonomický způsob poskytování hlasových služeb doplněných o řadu služeb nástavbových, které by na starších sítích nebyly realizovatelné. Mimo nesporných výhod se však VoIP komunikace i dnes pojí s řadou technických problémů, které je nutno řešit již při návrhu komunikační sítě. Jedním z těchto problémů je průchodnost vlastního hlasového/video toku, neboť porty pro tuto komunikaci jsou alokovány dynamicky. V lokálních sítích je tento problém relativně zvládnutý a jedna z nejrozšířenějších pobočkových ústředen současnosti - PBX Asterisk - nabízí řadu konfiguračních možností pro jeho zvládnutí. Jiná je situace na poli páteřních SIP serverů, jež obvykle fungují v režimu SIP proxy a audio/video vůbec neřeší. Zjištění vlastností a možností dostupných open-source řešení pro předávání audio/video toků i na úrovni páteřních sítí je hlavním cílem této bakalářské práce.

Zásady pro vypracování:
1. Student si nastuduje problematiku SIP komunikace a popíše ji.
2. Student identifikuje a popíše možnosti, výhody a nevýhody využití tzv. RTP proxy.
3. Student provede rešerši dostupných open-source řešení a vhodnosti jejich nasazení.
4. Student navrhne a realizuje testovací prostředí pro SIP proxy Kamailio.
5. Student změří a porovná možnosti dostupných open-source řešení pro předávání audia/videa.
6. Student provede shrnutí zjištěných poznatků.

Seznam doporučené odborné literatury:

[1] LEONARD, Nicholas. Understood SIP and IP Telephony in 3 days: For the beginner. Vydáno nezávisle, 2017. ISBN 978-1522084815.
[2] WALLINGFORD, Ted. Switching to VoIP. Sebastopol, CA: O'Reilly, 2005. ISBN 978-0596008680
[3] HARTPENCE, Bruce. Packet guide to Voice over IP. Tokyo: O'Reilly, 2013. ISBN 978-1449339678.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Rozhon, Ph.D.**

Datum zadání: 01.09.2018

Datum odevzdání: 30.04.2019

prof. Ing. Miroslav Vozňák, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 29. dubna 2019 .....

Na tomto místě bych rád poděkoval Ing. Janu Rozhonovi, Ph.D. za odbornou pomoc, vřelý přístup a čas, který mi věnoval během vypracovávání této práce.

Abstrakt

Cílem této bakalářské práce je realizovat testovací prostředí pro SIP Proxy Kamailio s funkčností rozšířenou pomocí volně dostupných řešení, která jsou určena pro předávání mediálního toků. Vliv jednotlivých řešení na výkonnost SIP Proxy serveru je následně otestován a porovnán. Nejprve se práce zabývá obecnými vlastnostmi technologie VoIP, zejména protokoly, které tato technologie využívá. Práce rovněž popisuje architekturu SIP serveru Kamailio a vlastnosti RTP proxy serverů.

Klíčová slova: VoIP, SIP, RTP, Kamailio, RTPProxy, RTPEngine, MediaProxy

Abstract

The aim of this bachelor thesis is to realize testing environment for SIP Proxy Kamailio with functionality enhanced by a freely available solutions, which are designed for proxying of media flow. Impact of these solutions on the performance of SIP Proxy server is then tested and compared. At first, the thesis deals with general properties of VoIP technology, especially the protocols, which are used by this technology. The thesis also describes architecture of the SIP server Kamailio and features of RTP proxy servers.

Key Words: VoIP, SIP, RTP, Kamailio, RTPProxy, RTPEngine, MediaProxy

Obsah

Seznam použitých zkratek a symbolů

ACELP	–	Algebraic Code Excited Linear Prediction
API	–	Application Programming Interface
B2BUA	–	Back to Back User Agent
DNS	–	Domain Name System
GB	–	Gigabyte
GPL	–	General Public License
HTTP	–	Hypertext Transfer Protocol
Hz	–	Hertz
ICMP	–	Intrnet Control Message Protocol
IP	–	Internet Protocol
ISO	–	International Organization for Standardization
ITU-T	–	International Telecommunication Unit - Telecommunication Standardisation Sector
kb/s	–	kilobit za sekundu
kB	–	kilobyte
LTS	–	Long Term Support
MI	–	Management Interface
ms	–	milisekund
NAPTR	–	Name Authority Pointer
NAT	–	Network Address Translation
OSI	–	Open Systems Interconnection
pcap	–	packet capturing
PCM	–	Pulse Code Modulation
PSTN	–	Public Switched Telephone Network
QoS	–	Quality of Service
RAM	–	Random Access Memory
RFC	–	Request for Comments
RPC	–	Remote Procedure Call
RTP	–	Real-time Transport Protocol
RTCP	–	Real Time Control Protocol
SCTP	–	Stream Control Transmission Protocol
SDP	–	Session Description Protocol
SER	–	SIP Express Router
SIP	–	Session Initiation Protocol
SRV	–	Service record
TCP	–	Transmission Control Protocol

TLS	–	Transport Layer Security
UA	–	User Agent
UAC	–	User Agent Client
UAS	–	User Agent Server
UDP	–	User Datagram Protocol
URI	–	Uniform Resource Identifier
VoIP	–	Voice over Internet Protocol
XML	–	eXtensible Markup Language

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

VoIP představuje moderní a ekonomicky výhodné řešení pro poskytování hlasových služeb skrze sítě využívající IP protokol. Existují však oblasti, ve kterých je nasazení této technologie spojeno s řadou problému. Jednou z těchto oblastní jsou sítě využívající NAT. Problém těchto sítí je spojen zejména s průchodností samotného hlasového toku. Jednou z možností řešící tento problém je nasazení takzvaných RTP proxy serverů.

První část této bakalářské práce se zabývá obecným popisem a vlastnostmi technologie VoIP. Je zde obsažen popis jednotlivých protokolů, které VoIP využívá, přičemž důraz je kladen na popis signalizačního protokolu SIP. Tato část dále obsahuje podrobný popis SIP serveru Kamailio. Následuje popis vlastností RTP proxy serverů a volně dostupných řešení plnících funkci RTP proxy, která jsou určena pro spolupráci se SIP serverem Kamailio.

Druhá část práce obsahuje podrobný popis instalace a konfigurace SIP serveru Kamailio. Následuje instalace volně dostupných řešení pro předávání RTP toků a jejich integrace se SIP serverem Kamailio. V závěrečné pasáži je provedeno testování těchto řešení a vyhodnocení jejich vlivu na vytížení serveru.

2 VoIP

VoIP (Voice over IP) je technologie, která pro přenos hlasu využívá datové sítě používající protokol IP (Internet Protocol). Rychlost, s jakou se datové sítě aktuálně rozvíjí, představuje pro VoIP možnost, jak se stát náhradou za klasickou veřejnou telefonní síť.

Veřejná telefonní síť, anglicky public switched telephone network (PSTN), je síť, určená k poskytování hlasových služeb, a to jak v digitální, tak i analogové podobě. Princip fungování této sítě spočívá v přepojování okruhů, to znamená, že na začátku každého hovoru je vytvořen mezi volajícími kanál, skrze který prochází samotný hovor. Tento kanál má v celé délce pevně vyhrazenou kapacitu. To je jeden z důvodů, proč je kvalita hovorů v PSTN velmi vysoká a téměř konstantní. V ohledu na přenosové vlastnosti sítě je ovšem tento princip velmi neekonomický, jelikož vyhrazená kapacita je ve chvílích, kdy volající nehovoří zcela nevyužita.[1]

VoIP využívá pro přenos hlasu datové sítě, jejichž prvotní účel byl přenos dat. V porovnání s PSTN pracují datové sítě využívající IP protokol na zcela jiném principu, zvaném jako přepojování paketů. Tento princip rozděluje vstupní data na menší části zvané pakety, ty jsou následně přenášeny sítí nezávisle na sobě. Mezi významné klady VoIP patří zejména ekonomická úspora, protože využíváním sítí, prvotně určených jen pro přenos dat odpadá nutnost výstavby nových telefonních okruhů. Dalším benefitem je administrace takovéto sítě, a to z důvodu, že hlasové i datové služby mohou být spravovány souběžně.[1][2][3]

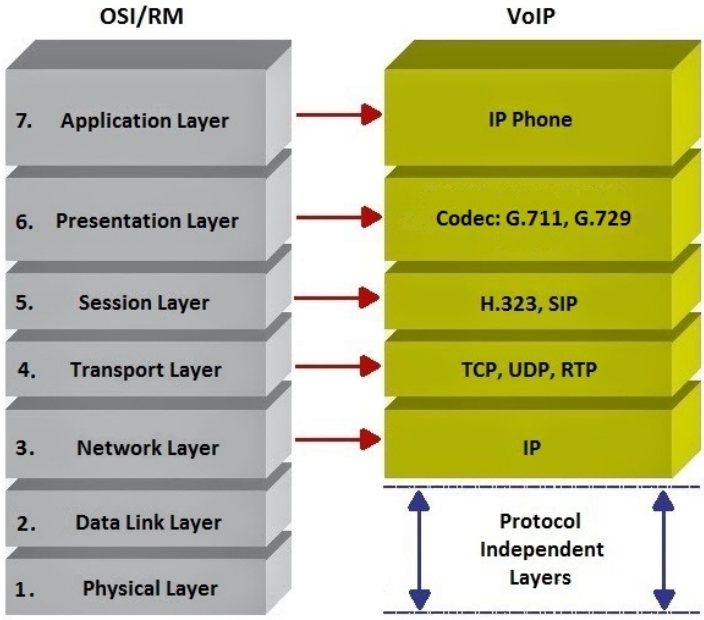
Nasazení hlasových služeb na datové sítě má však i své zápory. Tyto sítě nebyly navrženy pro přenos hlasu nebo obrazu v reálném čase. Při zavádění technologie VoIP se bylo nutné vypořádat se třemi základními problémy. Mezi tyto problémy patří:

- zpoždění (latency),
- nerovnoměrnost zpoždění (jitter),
- ztrátovost paketů (packet loss).

Zpoždění představuje dobu, kterou zabere přenos hlasu od volajícího k volanému. Pokud je tato hodnota vysoká, dochází k výraznému poklesu kvality hovoru. Jako jeden ze zdrojů zpoždění lze uvést směrování paketů. Směrování je proces, ve kterém směrovač na trase odesílatel-příjemce pracuje s informacemi obsaženými v hlavičkách IP paketů. Na základě získaných informací poté určí, jakým směrem data poslat. Proces směrování se může mnohokrát opakovat, v součtu proto tento zdroj zpoždění hraje poměrně velkou roli. Dle ITU-T by zpoždění v jednom směru nemělo přesáhnout hodnotu 150 ms, která je mezní pro zajištění hovoru o vysoké kvalitě.[2][3]

Jitter lze popsat jako nerovnoměrnost zpoždění mezi jednotlivými příchozími pakety. Tato nerovnoměrnost může být způsobena různými trasami, kterými pakety v síti putují, ale také změnami ve vytížení linek. Tento problém se řeší pomocí takzvaného jitter bufferu. V bufferu jsou přijaté pakety pozdrženy na dobu potřebnou pro to, aby se nerovnoměrnost zpoždění co nejvíce eliminovala. Pakety jsou následně posílány v pravidelných intervalech příjemci. Funkce bufferu sice minimalizuje nerovnoměrné rozestupy mezi pakety, ale za cenu dalšího, uměle vloženého zpoždění. Velikost bufferu proto musí být optimalizována v ohledu na celkové zpoždění.[1][2][3]

Ztrátovost paketů je poslední z hlavních problémů, vyskytující se například při přetížení sítě. Ztrátovost je ovšem způsobena i protokoly, které VoIP využívá, protože tyto protokoly negarantují doručení dat příjemci. V praxi by ztrátovost paketů neměla přesáhnout 1%. Kodeky, které jsou ve VoIP používány pro konverzi lidského hlasu do digitální podoby mají vůči ztrátovosti různé tolerance.[2]



Obrázek 1: VoIP z pohledu referenčního modelu OSI[4]

Výše uvedeným problémům spojených s VoIP se nelze vyhnout, je však možné je korigovat a minimalizovat. Jednou z možností je využití nástrojů QoS (Quality of service), které jsou schopny prioritizovat data citlivá například na zpoždění. VoIP pod sebou zahrnuje celou řadu protokolů a prvků, definujících chování této technologie a tím i její výhody/nevýhody. Ve zbývající části této kapitoly budou jednotlivé protokoly popsány z pohledu referenčního modelu OSI, viz obrázek 1. Tento model, který vypracovala organizace ISO (International Organization for Standardization), rozděluje síťovou komunikaci do sedmi vrstev, přičemž každá vrstva definuje protokol/protokoly specifické pro její účel a činnost.

2.1 IP protokol

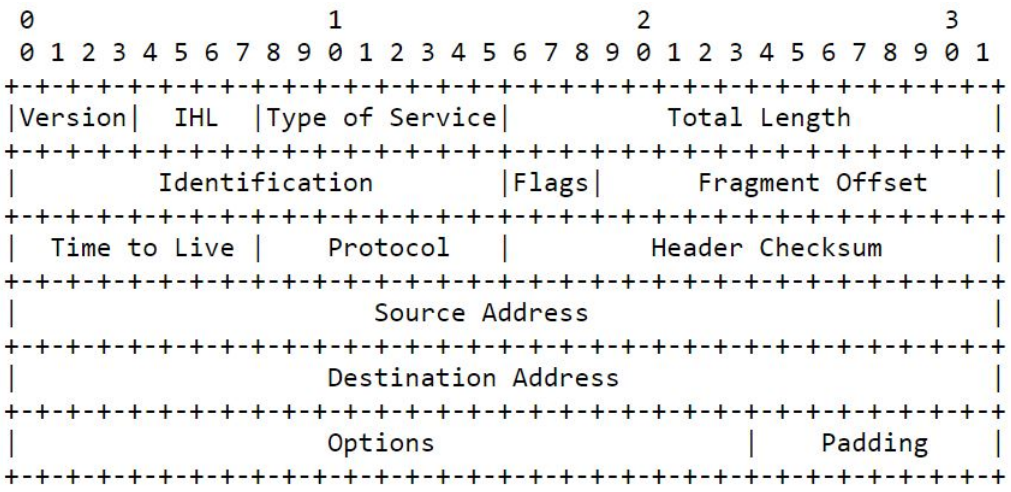
Jak již bylo zmíněno na začátku této práce, VoIP používá pro přenos dat sítě využívající IP protokol, který se dle referenčního modelu OSI nachází na jeho třetí vrstvě, viz obrázek 1. Úkolem tohoto protokolu je směrování paketů napříč sítěmi. Tuto činnost provádí metodou *best-effort* neboli metodou maximálního úsilí. To v praxi znamená, že se pokusí doručit data příjemci, ale v případě porušení či ztráty paketu nenabízí žádnou funkci, která by zajistila jeho znovu poslání.[5]

IP je nespojově orientovaný protokol. Před samotnou výměnou dat mezi odesílatelem a příjemcem se tedy nevytváří spojení. Pakety jsou v takovéto sítí směrovány nezávisle na sobě, je proto nutné, aby každý paket obsahoval položku identifikující příjemce. Tato informace je obsažená v každé hlavičce IP paketu a nazývá se IP adresa. IP adresa stejně jako celý protokol existuje ve dvou verzích, a to IPv4 a IPv6. Testovací topologie této práce využívá protokol IPv4, proto budou dále popsány vlastnosti čtvrté verze.[5][6]

IP adresa představuje jednoznačný identifikátor síťového rozhraní, na základě kterého je směrovač schopen poznat, do které sítě a také jakému konkrétnímu zařízení v dané síti paket náleží. U protokolu IPv4 má IP adresa délku 32 bitů, ale obvykle bývá vyjadřována v desítkové soustavě pomocí čtyř čísel oddělených tečkou. Z toho vyplývá, že každé číslo má délku 8 bitů, může tedy nabývat hodnot 0-255. Každý paket obsahuje dvě IP adresy, zdrojovou IP adresu, která specifikuje odesílatele a cílovou IP adresu, která určuje příjemce. Původní myšlenka protokolu IPv4, aby každé síťové zařízení komunikující ve veřejném internetu bylo možné jednoznačně identifikovat IP adresou však ztroskotalo na nedostatečném adresním prostoru, který tento protokol nabízí. V dnešní době proto velká část koncových zařízení, používajících IP protokol čtvrté verze disponuje privátní IP adresou. Níže jsou uvedeny rozsahy privátních IP adres:[5][6]

- 10.0.0.0 - 10.255.255.255
- 172.16.0.0 - 172.31.255.255
- 192.168.0.0 - 192.168.255.255

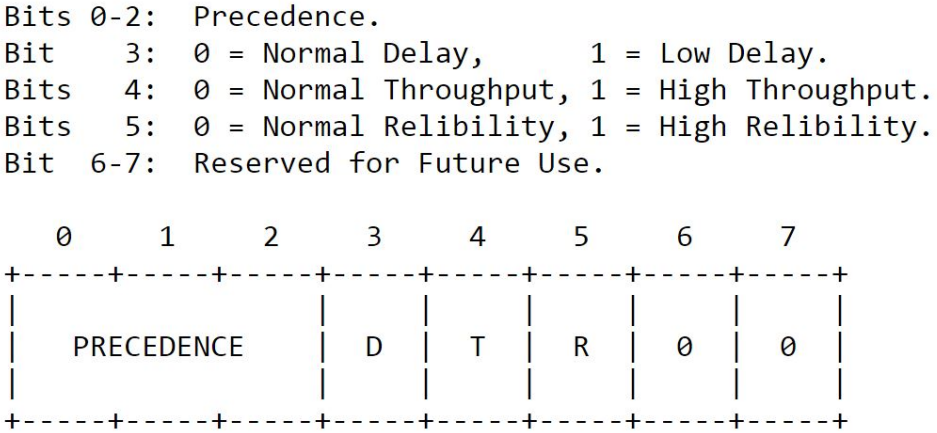
Využívání těchto adres však skrývá problém v tom, že jsou volně k dispozici jakémukoliv uživateli. To však představuje možnost výskytu dvou zařízení, používajících stejnou IP adresu. V praxi jsou proto privátní adresy používány pouze v rámci lokální sítě, ve které se uživatel nachází. Pokud chce komunikovat s veřejným internetem, musí být jeho privátní IP adresa nahrazena adresou veřejnou. Tato funkce, zvaná jako NAT (Network Address Translation) se obvykle provádí na hraničním směrovači, který se nachází mezi lokální sítí a sítí veřejnou. Zjednodušeně řečeno tento směrovač přepíše privátní IP adresu obsaženou v hlavičce IP paketu a nahradí ji adresou veřejnou, kterou sám disponuje. Směrovač provádějící NAT si poté udržuje záznam o tom, kterému uživateli danou adresu propůjčil, aby mohl zpětně určit, komu příchozí pakety náleží.[6]



Obrázek 2: Obsah IP záhlaví[5]

Na obrázku 2 lze vidět obsah IP záhlaví. Z pohledu VoIP je nejzajímavější položka Type of Service, která má délku osm bitů a jejím hlavním úkolem je informovat, že se jedná o paket vyžadující zvláštní zacházení.

Obrázek číslo 3 popisuje strukturu položky type of service. Například čtvrtý bit informuje, zda jsou data obsažená v paketu citlivá na zpoždění. Pokud tento bit nabývá hodnoty 1, je s paketem zacházeno tak, aby se během zpracovávání co nejméně zdržel. To může být vhodné pro real-time data, která jsou na zpoždění náchylná. V praxi však ne všechny směrovače znají postup, jak s daty vyžadujícími určitý typ priority zacházet. Obvykle proto tato položka nabývá hodnoty 0.[5]



Obrázek 3: Struktura položky Type of Service[7]

2.2 Transportní protokoly

V předchozí kapitole byl zmíněn IP protokol jakožto protokol síťové vrstvy, který je za pomocí IP adres schopen rozpoznat jednotlivé síťové rozhraní. V praxi však na jednotlivých rozhraních může komunikovat více aplikací souběžně. Na koncovém zařízení je proto nezbytné přesně určit, pro kterou z aplikací jsou konkrétní data určena. Pro tento úkol jsou určeny právě transportní protokoly, které jednotlivé aplikace definují pomocí takzvaných portů. Čísla portů jsou obsažena v záhlaví segmentu/datagramu, což jsou datové jednotky odpovídající transportní vrstvě. Tyto jednotky jsou dále zapouzdřeny do paketu, který k němu připojí své záhlaví. Na základě IP adresy obsažené v záhlaví IP paketu se data dostanou na správné síťové zařízení, které data zpracuje a následně dle čísla cílového portu určí aplikaci, pro kterou jsou přijatá data určena.

Každé záhlaví segmentu/datagramu obsahuje položku pro zdrojový a cílový port. Obsah každé z těchto položek obsahuje číslo o velikosti 16 bitů. To znamená, že tyto položky mohou obsahovat čísla v rozsahu od 0 do 65535. Tento rozsah se dělí do následujících třech skupin:[6][8]

- Dobře známé (Well-known): 0 - 1023
- Registrované (Registered): 1024 - 49151
- Dynamické (Dynamic): 49152 - 65535

První z výše uvedených skupin obsahuje čísla portů pro nejčastěji využívané aplikace, jako například HTTP (Hypertext Transfer Protocol) na portu číslo 80. Do druhé skupiny portů můžeme zařadit v této práci používaný SIP (Session Initiation Protocol) s porty číslo 5060, 5061. Nejpoužívanější skupinou z hlediska aplikací pracujících na zařízeních koncových uživatelů jsou dynamické porty, které nejsou pevně přiřazeny žádné aplikaci, ale jsou přidělovány dynamicky na začátku komunikace.[6]

Transportní vrstva obsahuje dva základní protokoly, jsou to protokoly TCP (Transmission Control Protocol) a UDP (User Datagram Protocol). Oba protokoly se z pohledu modelu OSI nachází na čtvrté vrstvě, viz obrázek 1.[6]

2.2.1 TCP

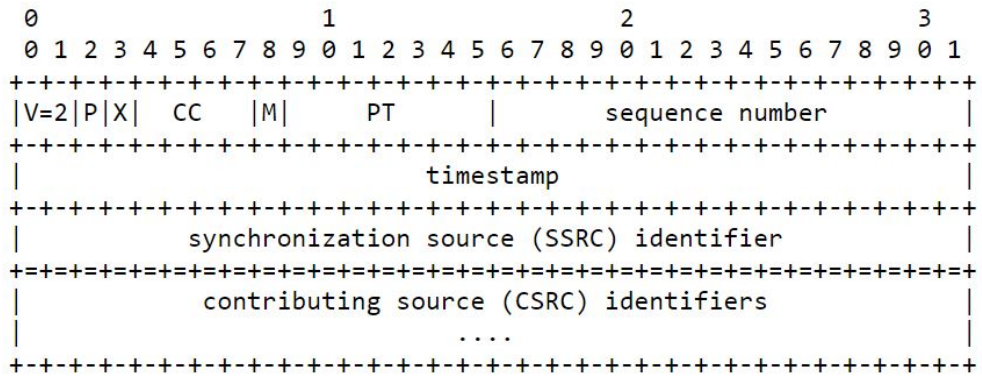
TCP je spolehlivý, spojově orientovaný protokol. To znamená, že před samotným přenosem dat se s druhou stranou naváže spojení, a to pomocí metody *Three-way handshake*. TCP je vhodný zejména pro aplikace požadující spolehlivé přenos garantující doručení dat a správnost pořadí. V případě ztráty nebo porušení paketu je paket zaslán znovu. Díky svým vlastnostem není protokol TCP vhodný pro přenos real-time dat, ale je využíván zejména aplikacemi, vyžadujícími spolehlivé doručení každého paketu.[8][9]

2.2.2 UDP

Protikladem TCP je protokol UDP. Tento protokol je nespolehlivý a nespojově orientovaný. To značí, že jsou data odesílána bez předem sestaveného spojení, ve kterém není zaručeno doručení dat do cíle ani jejich správnost či pořadí. Jednoduchost tohoto protokolu ho předurčuje pro nasazení v aplikacích, ve kterých je kladen důraz na rychlost a kde případná ztráta některého z paketů nebude mít zásadní vliv na celkový chod služby. Z pohledu VoIP je proto UDP protokol vhodný pro přenos real-time dat, která jsou citlivá na zpoždění.[8]

2.2.3 RTP

RTP (Real-time Transport Protocol) protokol je určený pro přenos hlasu či videa v reálném čase. RTP nelze jednoznačně definovat jako protokol transportní vrstvy, jelikož pracuje nad protokolem UDP. Obsah jeho záhlaví však nese znaky transportního protokolu, proto jsem se rozhodl RTP protokol zařadit do této kapitoly. Jeho hlavním úkolem je vylepšit a rozšířit vlastnosti UDP pro přenos real-time dat. Přináší zejména schopnost detekce ztracených paketů.[3][10]



Obrázek 4: Obsah RTP záhlaví[3]

Na obrázku 4 lze vidět obsah RTP záhlaví, které má obvykle velikost 96 bitů. Z prvků obsažených v záhlaví lze vyzdvihnout zejména tyto:

- Payload Type - položka definující formát užitečných dat, obsažených v paketu. Díky této položce je příjemce schopen určit pomocí jakého kodeku byla data zpracována.
- Sequence Number - položka o velikosti 16 bitů, jejíž hodnota se zvyšuje o jedna s každým odeslaným paketem. Sequence number hraje významnou roli při detekci ztráty paketů a jejich znovu uspořádání.
- Timestamp - představuje vzorkovací značku prvního oktetu RTP paketu. Hodnota je odvozena od lineárního časovače a je užitečná zejména pro zjištění hodnoty jitteru.[10]

Spolu s RTP je specifikován dohledový protokol RTCP (Real Time Control Protocol), jehož úkolem je sběr statistických informací týkajících se přenosu a jeho kvality.[3]

2.3 Signalizační protokoly

Signalizační protokoly hrají ve VoIP jednu ze zásadních rolí. Jejich úkolem je sestavení, udržování a ukončování spojení mezi volajícími stranami. Při pohledu na obrázek číslo 1 odpovídají tyto protokoly relační, čili páté vrstvě referenčního modelu OSI. Mezi nejznámější zástupce těchto protokolů patří jednoznačně protokoly H.323 a SIP. H.323 je protokol, který stál u počátků VoIP. Jeho první verze byla vydána již v roce 1996. Dnes je tento protokol spíše na ústupu a ve většině případů bývá nahrazován právě protokolem SIP. To je hlavní důvod, proč jsem se v této kapitole rozhodl věnovat pouze druhému ze zmíněných protokolů.[10]

2.3.1 SIP

SIP je textově orientovaný protokol, který byl vyvinut skupinou IETF (Internet Engeneering Task Force). Dnes je dostupný ve verzi 2, která je specifikovaná v RFC (Request For Comments) 3261-3265. Jedná se o takzvaný end-to-end protokol, což znamená, že koncová zařízení disponují veškerou potřebnou logikou pro sestavování, udržování a ukončování hovorů. Z hlediska protokolu nižší vrstvy může SIP spolupracovat s protokoly UDP i TCP.[2][10]

V SIP síti se můžeme setkat s těmito prvky:

- UA (User Agent) - jedná se o koncové zařízení, které vytváří/zpracovává spojení za účelem komunikace. V tomto případě se prvek odesílající žádosti nazývá UAC (User Agent Client). Prvek přijímající žádosti od UAC a generující odpovědi se nazývá UAS (User Agent Server).
- B2BUA (Back to Back User Agent)- je prvek, přijímající žádosti od UA. Přijaté žádosti následně upraví a odešle cílovému uživateli jako žádosti nové. Vytváří tedy dvě samostatné relace, u kterých hraje na jedné straně roli volaného a na straně druhé volajícího.
- SIP proxy - je server, který přijímá požadavky na sestavení hovorů od UA a tyto požadavky směruje dál k cíli. Na rozdíl od B2BUA nevytváří nové spojení. V praxi obvykle SIP proxy server spravuje určitou doménu. Jakmile chce klient spadající do určité domény vytvořit spojení s klientem, který se nachází v doméně jiné, pošle žádost na pro něj definovanou SIP proxy, ta dle svých záznamů či DNS (Domain Name System) vyhledá SIP proxy server volaného a následně na něj tuto žádost přepošle.
- SIP Registar - přijímá registrace od UA a aktualizuje informace v lokalizačním serveru, jehož informace mohou být následně využity například SIP proxy serverem.
- SIP Redirect - Prvek jenž přijímá požadavky, na základě kterých provede v lokalizační databázi vyhledání uživatele, jemuž je požadavek určen. Na rozdíl od proxy serveru požadavek nepřeposílá, ale pouze informuje jeho strůjce o aktuální adrese, na které může adresáta zastihnout.

- SIP Location - server obsahující databázi, která nese informace o umístění UA nebo SIP proxy serverů.[6][10]

SIP identifikuje jednotlivé koncové uživatele pomocí SIP URI (Uniform Resource Identifier), která se skládá z části definující uživatele, za kterou následuje @ a poté část definující doménu, pod kterou tento uživatel spadá. URI může být rovněž doplněna o dodatečné parametry, které se oddělují středníkem. SIP URI může vypadat například takto:

- sip:bob@mydomain.com
- sip:123456789@192.168.0.1:5060[6][10]

Samotná signalizace probíhá pomocí zpráv, které se dělí na dvě části, SIP žádosti (SIP Requests) a SIP odpovědi (SIP Responses).

Základní SIP žádosti používané v SIP jsou následující:

- INVITE - žádost o sestavení spojení.
- REGISTER - žádost o registraci uživatele, kterou posílá klient na server.
- ACK - žádost potvrzující přijetí konečné odpovědi na žádost INVITE.
- OPTIONS - žádost o zaslání informací, týkajících se funkcí serveru.
- BYE - žádost o ukončení spojení, které bylo úspěšně sestaveno.
- CANCEL - žádost o ukončení ještě nesestaveného spojení.[6][10]

SIP odpovědi jsou rozděleny do 6 tříd, přičemž každá třída má svůj význam a říká nám, o jaký typ odpovědi se jedná.

Třídy odpovědí jsou následující:

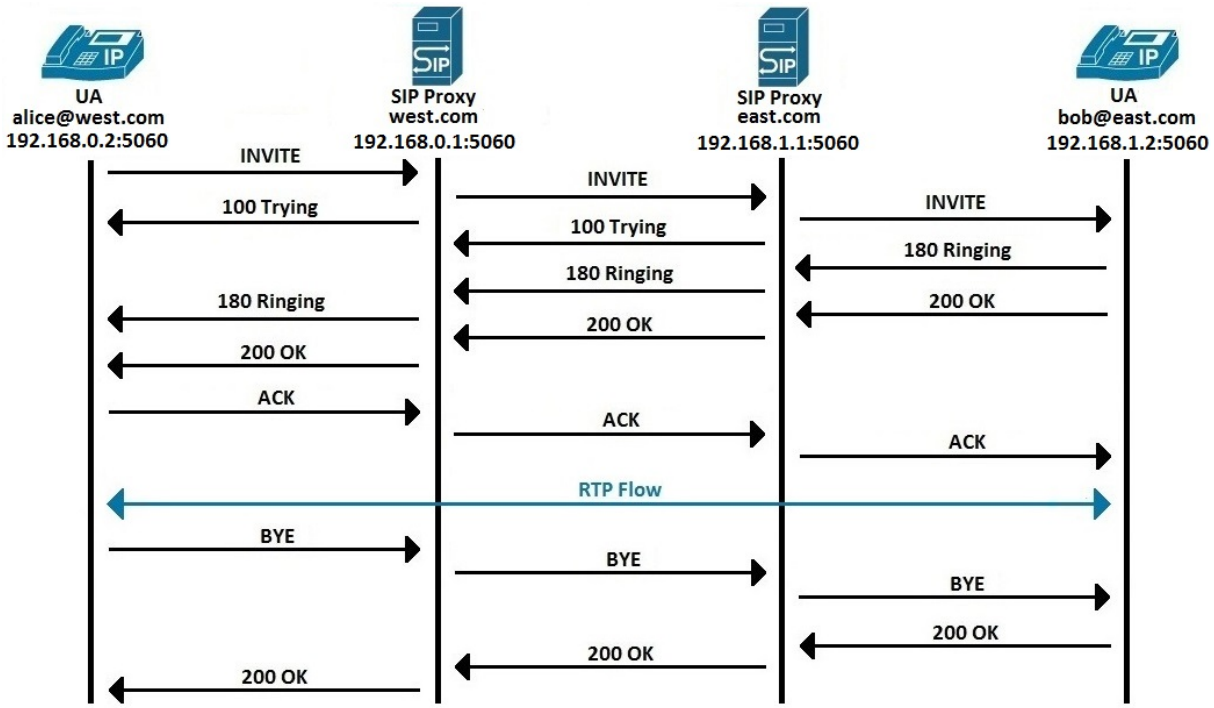
- 1xx - dočasné informativní odpovědi. Lze zde zařadit například odpověď 180 Ringing.
- 2xx - pozitivní finální odpovědi, typickým zástupcem je odpověď 200 OK.
- 3xx - odpovědi posílané serverem, které se týkají přesměrování. Příklad: 305 Use Proxy.
- 4xx - negativní finální odpovědi a asi nejznámější zástupce 404 Not Found.
- 5xx - odpovědi oznamující chybu na straně serveru, například 502 Bad Gateway.
- 6xx - odpovědi spojované s globální chybou, například 600 Busy Everywhere.[6][10]

Každá ze zmíněných SIP zpráv obsahuje v hlavičce řadu parametrů, popisujících například adresu příjemce či odesílatele. Níže jsou vypsány základní položky objevující se v SIP hlavičce:

- Request-URI - pole obsahující název žádosti, například INVITE a URI prvku, kterému je žádost určena.
- From - logická adresa volajícího.
- To - logická adresa volaného.
- Via - záznam trasy, kterou žádost putovala. Každý prvek na trase přidá tuto položku se svou fyzickou adresou. Při posílání odpovědi jsou tyto položky použity pro směrování.
- Record-Route - položka obsahující adresy SIP proxy serverů, které vyžadují své setrvání v signalizační trase po celou dobu jejího trvání.
- Route - pole obsahující adresy serverů, přes které bude žádost směrována.
- Call-ID - identifikátor hovoru.
- Cseq - položka přiřazující číslo každé SIP žádosti. Toto číslo se s každou žádostí zvyšuje. Příjemce je tak schopen poznat, zda se jedná například o znovu poslanou zprávu.
- Contact - fyzická adresa odesílatele, na které očekává příjem dalších zpráv.[6][10]

Dříve zmíněné SIP zprávy a parametry hlaviček v nich obsažených nám popisují pouze proces sestavení, či ukončení spojení mezi dvěma volajícími. Pro popis samotné relace využívá SIP takzvaný SDP (Session Description Protocol) protokol, který je posílán spolu se SIP zprávami INVITE a 200 OK. Základní položky obsažené v SDP jsou následující:

- v (protocol version) - verze protokolu. Vždy je nastavena na hodnotu 0.
- o (owner/creator) - označuje zakladatele relace.
- s (session name) - název relace.
- c (connection information) - informace o připojení. Například: IN IP4 192.168.0.1.
- t (time session starts and stops) - obsahuje čas začátku a konce relace.
- m (media information) - popisuje typ přenosu, dále port, na kterém se očekává příjem mediálních toku a na závěr typ užitečné zátěže (kodek).
- a (media atributes) - rozšiřující vlastnosti. Bývají zde popsány podporované kodeky.[6][10]



Obrázek 5: Ukázka SIP komunikace[27]

Před popisem komunikace, znázorněné na výše uvedeném obrázku[1], je vhodné zmínit typy SIP proxy serverů, se kterými se můžeme v praxi setkat. Typy SIP proxy serverů jsou následující:

- stateless (bezestavové),
- stateful (stavové).

Stateless SIP proxy jsou jednoduché servery, které jednotlivé SIP zprávy pouze přeposílají na základě jejich obsahu, ale neudržují si o nich žádné informace. Nejsou schopny poskytovat pokročilé funkce, jako například větvení hovorů či jejich přesměrování.

Stateful SIP proxy servery udržují záznam o žádostech či odpovědích, které obdržely. Nejčastěji se vyskytujícím typem SIP proxy je takzvaný transakční statefull SIP proxy, který si udržuje stavy k žádosti, a to do té doby, dokud není vyřízená. Druhým zástupcem stateful serverů jsou servery dialogové, které udržují stav dialogu (hovoru) dokud není ukončen. Jako příklad transakčního stateful SIP proxy serveru lze zmínit Kamailio, jehož instalace a konfigurace je součástí praktické části této práce.[6][10]

Na obrázku číslo 5 lze vidět SIP komunikaci mezi Alicí, jejíž doména je spravována SIP proxy serverem west.com a Bobem, jehož oblast spravuje SIP proxy east.com.

[1]Při tvorbě obrázků číslo 5, 6 a 15 byla použita sada volně dostupných ikon, viz reference číslo [21].

Alice posílá žádost INVITE na SIP proxy west.com. Spolu s INVITE posílá i SDP nesoucí informace, jako IP adresu a port, na kterém očekává příjem mediálních dat. Proxy west.com po přijetí INVITE okamžitě posílá zprávu 100 Trying, aby předešel znovu zasílání žádosti INVITE. Z *Request-URI* obsažené v INVITE zjistil, že se volající nenachází v doméně, kterou spravuje. Za pomocí DNS či statických záznamů proto provede vyhledávání adresy SIP proxy serveru east.com. Než zprávu přepošle, přidá do ní položku *Record-route*.

SIP proxy east.com přijal INVITE a posílá 100 Trying. Z *Request-URI* zjistil, že volaný spadá do jeho domény. Provede proto jeho vyhledání pomocí svých záznamů nebo Location serveru. Než zprávu INVITE přepošle, přidá do ní položku *Record-route*, stejně jako tomu učinil server west.com. Žádost INVITE dorazila Bobovi, ten posílá zprávu 180 Ringing, do které zkopíruje všechny *Record-route* položky, které obdržel v INVITE. Dále přidá položku *Contact*, nesoucí jeho adresu. Zpráva následně poputuje k Alici dle položek *Via*, které do žádosti INVITE přidal každý prvek, nacházející se na trase mezi Alicí a Bobem. Stejným způsobem jako zprávu 180 Ringing posílá Bob i finální odpověď 200 OK, s tím rozdílem, že k ní připojí i SDP. Ukázka položek *Via*, dle kterých jsou směrovány odpovědi 180 Ringing a 200 OK vypadá takto:

```
Via: SIP/2.0/UDP 192.168.1.1:5060
Via: SIP/2.0/UDP 192.168.0.1:5060
Via: SIP/2.0/UDP 192.168.0.2:5060
```

Alice, která přijala zprávy 180 Ringing a 200 OK posílá zprávu ACK, kterou utvrzuje sestavení spojení. Zpráva ACK bude směrována pomocí informací získaných z *Record-route*, které byly součástí odpovědí na INVITE. Tyto informace, ovšem v opačném pořadí (jelikož byly získány z odpovědi), vloží Alice do pole *Route*. Nutno dodat, že pro případ, ve kterém by SIP proxy nepoužívaly funkci *Record-route* by Alice i Bob směrovali zbylé zprávy pomocí pole *Contact*, které si v předchozích SIP zprávách vyměnili. To však neplatí pro náš příklad. Obsah položky *Route*, dle které Alice směruje zprávu ACK vypadá následovně:

```
Route: <sip:west.com;lr>, <sip:east.com;lr>
```

Alice posílá žádost dle prvního *URI* obsaženého v poli *Route*. Poté, co SIP proxy west.com příjme ACK, odstraní z pole *Route* jeho první položku a žádost odešle dle *URI*, které v položce *Route* zbývá. Na SIP Proxy east.com se proces opakuje, s tím rozdílem, že po odstranění poslední položky z pole *Route* odesílá zprávu dle *Request-URI*. Bob přijal ACK, čimž je proces sestavení hovoru dokončen, může tedy začít samotná konverzace, jejíž parametry byly specifikovány pomocí SDP.

Jakmile konverzace skončí, posílá Alice zprávu BYE, která bude odeslána pomocí stejného postupu jako zpráva ACK. Po přijetí zprávy BYE odpovídá Bob zprávou 200 OK, která není směrována dle položek *Route* a *Request-URI*, nýbrž pomocí hodnot získaných z polí *Via*. Poté, co Alice příjme zprávu 200 OK je hovor úspěšně ukončen.[10][11]

2.4 Kodeky

Kodeky představují nedílnou součást každého VoIP telefonu. Jejich primárním účelem je konverze lidského hlasu z analogové do digitální podoby. Z pohledu OSI modelu, viz obrázek 1, odpovídá funkce kodeku prezentační, čili šesté vrstvě. Každý z kodeků používá pro proces digitalizace odlišný algoritmus, je proto nezbytné, aby v rámci hovoru používali volající stejný kodek.

Mezi nejznámější algoritmus používaný pro digitalizaci hlasu patří PCM (Pulse Code Modulation). PCM je využíván kodekem G.711 a dělí proces digitalizace do následujících kroků:

- **Vzorkování** - je proces, během kterého jsou ze spojitého analogového signálu v pravidelných intervalech odebírány vzorky. Tento interval je specifikován vzorkovací frekvencí kodeku. V případě PCM je vzorkovací frekvence 8000 Hz. Vzorek je tedy odebrán každých 0,125 ms. Hodnota vzorkovací frekvence vychází z frekvence lidského hlasu. Většina zvuků, které vydáváme, se nachází v rozsahu 0-4000 Hz. Nyquistův teorém říká, že pro zajištění dobré reprodukce signálu je zapotřebí, aby vzorkovací frekvence byla minimálně dvakrát větší, než maximální frekvence vzorkovaného signálu. V našem případě tedy 8000 Hz.

- **Kvantování** - je proces, ve kterém jsou odebrané vzorky přiřazovány k určité kvantizační hladině, která specifikuje intenzitu hlasu. Každý vzorek je přiřazen k hladině, ke které má z pohledu intenzity nejblíže. V tomto kroku může docházet k takzvanému kvantizačnímu zkreslení, které nastává v případě, kdy intenzita odebraného vzorku přesně neodpovídá některé z hladin. Vzorek tedy musí být k nejbližší hladině uměle přiřazen, tím dochází k degradaci hlasu z pohledu jeho intenzity.

- **Kódování** - přiřazuje jednotlivým kvantovaným vzorkům binární hodnotu dle kvantizační hladiny, ke které byly přiřazeny. U PCM je těchto hladin celkem 256. Tuto hodnotu lze vyjádřit pomocí osmi bitového čísla, z čehož vyplývá, že každý vzorek má velikost právě osm bitů. Pokud tuto hodnotu vynásobíme vzorkovací frekvencí, dostaneme hodnotu 64 kb/s, což je přenosová rychlost algoritmu PCM.[2][3]

Ostatní kodeky používané ve VoIP obvykle využívají algoritmy vycházející právě z PCM. Hlavním důvodem pro použití odlišných technik digitalizace hlasu je zejména snaha zredukovat přenosovou rychlost, a to bez znatelného vlivu na kvalitu hlasu. V praktické části této práce budou použity kodeky G.711 a G.729, jejichž parametry jsou shrnuty v následující tabulce.

Kodek	**Algorimus**	**Přenosová rychlost [kb/s]**	**Vzorkovací frekvence [Hz]**	**Payload type**
G.711	PCM A-law	64	8000	8
G.729	CS-ACELP	8	8000	18

Tabulka 1: Specifikace použitých kodeků[2][3]

3 Kamailio

V předchozí kapitole jsme si představili základní vlastnosti a protokoly spojené s technologií VoIP. Některé z prvků SIP sítě, které byly zmíněny, mohou být implementovány pomocí softwaru Kamailio, jehož podrobným popisem se zabývá tato kapitola.

Kamailio je volně dostupný software, vydaný pod licencí GPL (General Public License), určený pro Unix/Linux systémy. Kořeny Kamailia sahají do roku 2001, kdy byl institutem FhG Fokus Research vytvořen projekt SIP Express Router (SER). V roce 2005 se od projektu SER odtrhla skupina, která svůj projekt nazvala OpenSER. Z důvodu porušování ochranných známek byl v roce 2008 projekt OpenSER přejmenován na Kamailio. Ještě v témže roce se projekty SER a Kamailio (OpenSER) opět sloučily, aby sjednotily svou práci a výhody, které jejich projekty nabízely. S příchodem verze 3.0.0 v roce 2010 bylo sjednocení dokončeno.[12][13]

Kamailio je napsáno v programovacím jazyce C. Jeho architektura je dimenzována pro maximální výkon, se schopností sestavovat tisíce hovorů každou sekundu. Díky malé hardwarové náročnosti je tento software vhodný jak pro nasazení v systémech s omezeným výkonem, tak v prostředí s velkou výpočetní kapacitou. Kamailio může být nasazeno jako:

- Proxy server,
- Redirect server,
- Registrar server,
- Location server,
- Application server.

Uživatel může do Kamailia naimplementovat celou řadu řešení, rozšiřujících jeho základní funkčnost. Tato funkce je nezbytná pro to, aby mohlo být Kamailio nasazeno v sítích, které mají specifické nároky. Rozšíření jsou pro Kamailio dostupná v podobě přídavných modulů, kterým bude věnován prostor v další části této kapitoly. Jako příklad rozšiřujících funkcí lze zmínit: vyvažování zátěže, průchod SIP a RTP dat skrze NAT, či WebSocket pro WebRTC.[13][14]

3.1 Architektura

Kamailio disponuje architekturu, kterou si může každý uživatel přizpůsobit dle účelu, pro které má být nasazena. Architektura Kamailia se dělí na tyto tři části:

- jádro,
- interní knihovny,
- moduly.

3.1.1 Jádro

Jádro Kamailia po znovu spojení s projektem SER prošlo určitými změnami. Některé z funkcí, které byly u starší verze součástí jádra, jsou nyní dostupné jako interní knihovny. Jádro obsahuje:

- Memory manager - prvek, jehož úkolem je zjednodušit práci se sdílenou a privátní pamětí tím, že poskytuje mezi pamětí a aplikací jednoduché rozhraní. Při inicializaci si vyhrazuje velkou část systémové paměti a následně aplikaci přiděluje její části dle potřeby.
- Locking system - vlastní uzamykací systém Kamailia. Jeho hlavním prvkem je *mutex* semafor. Zamykání může být řešeno i pomocí proměnné nebo pole zámků.
- SIP transport layer - implementuje podporu pro UDP, TCP, TLS, SCTP a přidává rovněž podporu pro záznamy NAPTR a SRV náležící službě DNS (Domain Name System).
- SIP message parser - Kamailio definuje vlastní parser, známý jako inkrementující. Tento parser prochází každou SIP zprávu, dokud nenalezne požadované informace nebo dokud nedojde na konec SIP zprávy.
- Configuration file parser and interpreter - pro parsování konfiguračního souboru využívá prvky *flex* a *bison*. Tyto prvky sestavují strom akcí, které mají být provedeny pro každou SIP zprávu.
- Control interface - představuje rozhraní určené pro komunikaci se SIP serverem za účelem jeho správy a administrace. Doporučeným rozhraním pro tyto účely je RPC (Remote Procedure Call). Další typ kontrolního rozhraní, který však není doporučen, se nazývá MI (Management Interface). [15]

3.1.2 Interní knihovny

Interní knihovny byly zavedeny s příchodem verze 3.0.0. Jedná se o sbírku funkcí, napsaných v jazyce C, které jsou automaticky načteny za běhu programu, a to v případě, kdy některý z modulů potřebuje využít jednu z funkcí, které knihovny nabízí. Interní knihovny obsahují komponenty, které byly přímou součástí staré verze jádra. Díky přesunutí některých funkcí z jádra do knihoven se jádro stalo jednodušší, a tím i stabilnější. Byly zde přesunuty prvky jako: Database API, Management Interface, Statistics Engine.[15]

3.1.3 Moduly

Moduly představují přídavný software, jehož úkolem je rozšířit základní funkčnost Kamailia. Jednotlivé moduly jsou ke Kamailiu připojovány pomocí takzvaného *module interface*. Nedochází proto k přímému propojení mezi moduly a samotným jádrem, což má kladný vliv zejména na celkovou stabilitu aplikace. Počet modulů se s každou verzí Kamailia rozšiřuje, nyní jich je k dispozici více než 150.[14]

3.2 Hlavní konfigurační soubor

Hlavní konfigurační soubor Kamailia nese název kamailio.cfg. Tento soubor umožnuje nastavení či specifikování celé řady parametrů, a tím i možnost upravovat chování serveru. Konfigurační soubor kamailio.cfg lze rozdělit na tyto sekce:

- sekce globálních parametrů,
- sekce nastavení modulů,
- sekce směrovacích bloků.

3.2.1 Sekce globálních parametrů

První část konfiguračního souboru obsahuje položky popisující parametry jádra a parametry nadefinované uživatelem. Hodnota těchto parametrů je obvykle reprezentována číslem, řetězcem znaků nebo boolean hodnotou (yes/no). Zde je ukázka globálních parametrů:[16]

```
disable_tcp=yes
alias="mydomain.com"
listen=udp:192.168.0.1:5060
```

3.2.2 Sekce nastavení modulů

Sekce umožňující uživateli definovat moduly, které mají být načteny a následně specifikovat jejich parametry. Tyto dvě činnosti jsou spouštěny pomocí klíčových slov "loadmodule" a "modparam", přičemž "modparam" je složen ze tří argumentů: název modulu, název parametru, hodnota parametru. Níže je uvedena ukázka načtení a nastavení parametrů modulu:[16]

```
loadmodule "debugger.so"
modparam("debugger", "cfgtrace", 1)
modparam("debugger", "log_level_name", "exec")
```

3.2.3 Sekce směrovacích bloků

Sekce obsahuje veškerou směrovací logiku a mechanismy, pomocí kterých Kamailio zpracovává každou SIP zprávu. Níže je uveden seznam směrovacích bloků:

- request_route - hlavní směrovací blok zpracovávající SIP žádosti.
- onreply_route - blok zpracovávající SIP odpovědi.
- failure_route - blok pro zpracování negativních SIP odpovědí.
- branch_route - blok obsahující souhrn akcí, které mají být provedeny pro každou žádost v rámci SIP transakce. Tato sekce se používá při větvení hovorů.[16]

4 RTP proxy a jeho dostupná řešení pro Kamailio

Ve výchozím stavu funguje Kamailio jako SIP proxy server. To znamená, že datový provoz jdoucí přes SIP proxy server je spojen pouze se SIP signalizací, ale audio/video, které si účastníci vyměňují, putuje sítí nezávisle na SIP proxy. Abychom Kamailiu umožnili fungovat v režimu RTP proxy, je zapotřebí využít některé z dostupných modulů zabývajících se touto problematikou.

Hlavní důvodem pro nasazení RTP proxy serverů je NAT, viz kapitola 2.1. Jak jsme si uvedli, SIP pracuje dle OSI modelu na jeho páté vrstvě. NAT však využívá třetí a čtvrtou vrstvu toho modelu. IP adresy, které jsou vloženy do SIP a SDP zpráv nebudou v tomto případě přeloženy. Klient, jenž obdrží SIP INVITE, nebude schopen odpovědět, jelikož položka *Via* bude obsahovat pouze privátní IP adresu volajícího. Stejný problém je spojen i s SDP zprávou, ve které volaný v položkách *c* a *o*, nalezne pouze privátní IP adresy, na kterých volající očekává příjem RTP paketů. Primárním účelem RTP proxy je tedy umožnit průchod RTP paketů skrze sítě využívající NAT. Nasazení RTP proxy serverů může být vhodné i pro jiné aplikace, jako například:[10][17]

- optimalizace datového toku,
- nahrávání hovorů,
- sběr informací o kvalitě hovorů,
- přehrávání přednahraných oznámení,
- nastavení ToS/QoS položek, viz kapitola 2.1,
- propojovací funkce mezi uživateli používajícími odlišné verze IP protokolu,
- transkodování, funkce používaná mezi účastníky/sítěmi vyžadujícími použití odlišného kodeku.[17][19][20]

Hlavní problém, který se pojí s nasazením RTP proxy serverů je jejich vliv na nárůst zpoždění mediálního toku. Data obsažená v RTP paketech obvykle putují přímo mezi volajícími stranami. V případě nasazení RTP proxy serverů putuje veškerý mediální tok na RTP proxy server. Ten následně určí, komu daný mediální tok náleží a teprve poté přeposílá data cílovému uživateli.

Ve zbylé části této kapitoly budou zmíněny řešení pro předávání RTP toků, která jsou dostupná pro Kamailio. Tyto řešení v podobě přídavných modulů povětšinou nabízí podobné funkce. Mohou se ale lišit kontrolním protokolem, pomocí kterého komunikují se SIP serverem a také programovacím jazykem, ve kterém byly napsány.

4.1 RTPProxy

RTPProxy, jehož autorem je Maxim Sobolyev, vznikl za účelem zpřístupnit technologií VoIP mezi klienty, kteří se v síti nacházejí za NAT. V současné době je tento software spravován společností Sippy Software, Inc.

Pro optimalizaci datového toku spolupracuje RTPProxy s modulem NATHELPER, který je součástí Kamailia. NATHELPER je schopen pracovat s více RTPProxy servery souběžně a vyvažovat tak mezi nimi zátěž dle aktuální potřeby. V případě výpadku některého z RTPProxy serverů umožňuje NATHELPER přesměrovat zátěž na server jiný.[17]

4.1.1 Princip fungování RTPProxy

Jakmile SIP proxy (Kamailio) obdrží žádost INVITE, vyjme z ní hodnotu položky *Call-ID* a pošle ji na RTPProxy skrze UNIX, nebo UDP soket. RTPProxy nahlédne do svých záznamů, zda relace s tímto *Call-ID* existuje, pokud ano, pošle UDP port, který této relaci odpovídá. Pokud *Call-ID* nenajde, vytvoří pro tuto relaci nový záznam a přidělí jí první volný UDP port z definovaného rozsahu. Následně pošle číslo tohoto portu zpět na Kamailio, které tento port spolu s IP adresou RTPProxy serveru použije pro přepsání SDP zprávy, konkrétně položky media_ip:port.

Poté, co Kamailio příjme pozitivní finální odpověď (200OK), která obsahuje SDP zprávu, vyjme z ní hodnotu *Call-ID* a pošle ji na RTPProxy. RTPProxy v tomto případě nevytváří nový záznam, ale pouze zkontroluje probíhající relace. Pokud v nich konkrétní *Call-ID* nalezne, vrátí Kamailiu UDP port. Tentokrát se jedná o port pro druhou stranu účastnící se hovoru. Pokud relaci s tímto *Call-ID* nenalezne, pošle SIP proxy serveru chybové hlášení. Jakmile Kamailio obdrží číslo portu, opět ho spolu s IP adresou RTPProxy serveru použije pro přepsání položek *media_ip:port* tak, aby odkazovaly právě na RTPProxy server.

Jakmile je vytvořeno spojení mezi volajícími stranami, naslouchá RTPProxy na portech, které pro tento hovor vyhradil a očekává přijetí alespoň jednoho UDP paketu od každé strany účastnící se hovoru. Poté, co tyto pakety příjme, použije RTPProxy jejich zdrojové IP adresy a čísla portů pro vyplnění svých záznamů. Po zbytek hovoru tyto záznamy využívá pro určení IP adresy a portu, na kterých účastníci hovoru očekávají příjem RTP paketů.[18]

4.2 RTPEngine

RTPEngine představuje další volně dostupný software spolupracující s Kamailiem, jehož úkolem je předávání RTP toků. Tento software původně nesoucí název Mediaproxy-ng je aktivně spravován firmou Sipwise a je určen jako náhrada za ostatní dostupné RTP proxy servery.

RTPEngine na rozdíl od ostatních RTP proxy serverů nabízí funkci *In-kernel packet forwarding*. Smyslem této funkce je redukce zátěže CPU a minimalizace zpoždění mediálních dat.

Obvykle při zpracování paketu dochází nejprve k jeho průchodu skrze síťovou část kernelu a až poté dochází ke zpracování dat samotnou aplikací, které jsou data určena. Každý paket tedy prochází mezi prostředím kernelu a uživatelským prostředím. RTPEngine disponuje dvěma komponentami, díky kterým je schopen se tomuto postupu vyhnout. První z nich je vlastní kernel modul, který zajistí zpracování dat pouze v prostředí kernelu. Druhým prvkem je rozšíření pro iptables, které slouží pro přesměrování veškerého mediálního toku do kernel modulu.[19]

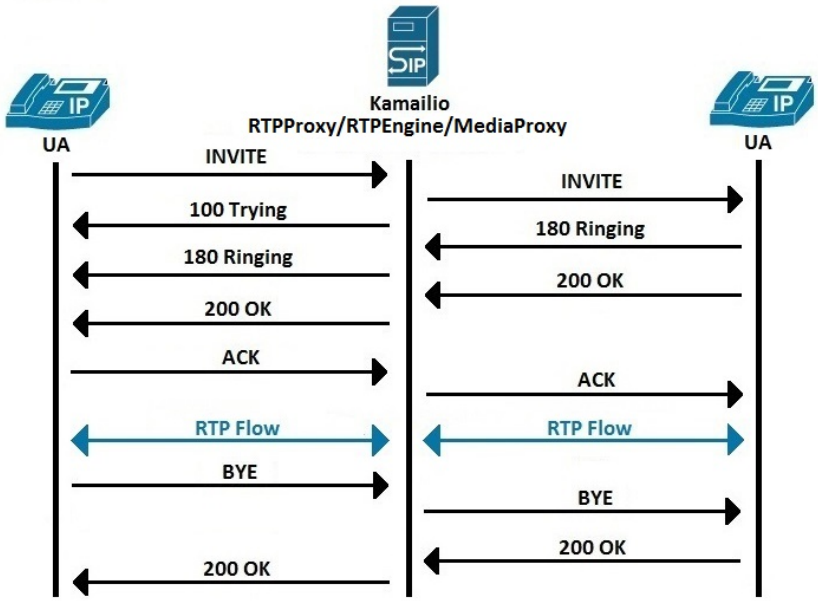
4.3 MediaProxy

MediaProxy, stejně jako RTPProxy a RTPEngine, představuje takzvané far-end řešení pro překonání NAT. Architektura MediaProxy se skládá ze dvou základních komponent:

- MediaProxy-dispatcher,
- MediaProxy-relay.

MediaProxy-dispatcher pracuje na stejném zařízení jako SIP proxy. Jeho úkolem je vybrat pro hovor některý z dostupných MediaProxy-relay serverů.

MediaProxy-relay může fungovat na stejném zařízení jako SIP proxy, ale klidně i jako vzdálený server. Podmínkou je, aby disponoval veřejnou IP adresou. Úkolem relay serveru je samotné předávání RTP toků mezi volícími. Architektura MediaProxy počítá s nasazením většího množství relay serverů, které tak mezi sebou budou schopny vyvažovat zátěž. Pokud je k dispozici pouze jeden relay server, je možné, aby se SIP proxy komunikoval přímo bez pomoci dispečera.[20]



Obrázek 6: SIP komunikace s použitím Kamailia a modulů pro předávání RTP toků [27]

5 Instalace a konfigurace

Hlavní náplní této práce bylo otestovat dostupná řešení pro předávání RTP toků, která jsou k dispozici pro SIP Proxy Kamailio. Nejprve bylo nutné nainstalovat samotný SIP proxy server. Kamailio lze provozovat pouze na unixových/linuxových operačních systémech, bylo proto nezbytné zajistit zařízení s takovýmto operačním systémem. Vedoucím práce mi byl pro tyto účely vytvořen virtuální server s operačním systémem Ubuntu 18.04 LTS - 64bit, který byl dostupný pod IP adresou 158.196.244.195. Veškerá instalace a konfigurace, která je zmíněná v této kapitole probíhala po přihlášení se jako superuživatel *root*.

5.1 Kamailio

5.1.1 Kamailio - Instalace

První věcí, kterou jsem vykonal ještě před samotnou instalací Kamailia, bylo nainstalování databázového serveru. Ten je potřebný pro správu uživatelů, jejich přidávání či odebírání. Kamailio nedisponuje vlastní databází, ale spolupracuje například s databází MySQL. Já jsem zvolil databázi MariaDB, kterou jsem nainstaloval následujícími příkazy:

```
1. apt-get install software-properties-common
2. apt-key adv --recv-keys --keyserver
   hkp://keyserver.ubuntu.com:80 0xF1656F24C74CD1D8
3. add-apt-repository 'deb [arch=amd64]
   http://mirror.zol.co.zw/mariadb/repo/10.3/ubuntu bionic main'
4. apt update
5. apt install mariadb-server
```

Prvním příkazem jsem nainstaloval rozšíření, umožňující snadněji pracovat se souborem obsahujícím seznam balíčků, který se nachází v */etc/apt/source.list*. V dalším kroku jsem stáhnul klíč, který je potřebný pro instalaci databáze. Následuje příkaz, ve kterém jsem přidal záznam do seznamu balíčků, který informuje, kde se balíček nachází. Pomocí kroku čtyři jsem tento seznam aktualizoval. Posledním příkazem jsem provedl samotnou instalaci databáze, v jejímž průběhu jsem byl vyzván k zadání hesla, které bude sloužit uživateli *root* pro připojení do databáze.[22]

Tímto je databázový server nainstalován a připraven. Nyní můžeme provést instalaci samotného Kamailia. Pro mojí verzi operačního systému byla v Ubuntu repozitářích dostupná jedna z posledních verzí Kamailia, konkrétně verze 5.1.2. Nebylo proto nutné přidávat do seznamu balíčků záznam, jako tomu bylo v případě instalace databázového serveru. Spolu s Kamailiem jsem zároveň nainstaloval MySQL modul, jehož úkolem je vytvořit rozhraní mezi Kamailiem a databázovým serverem. Samotnou instalaci jsem provedl následujícím příkazem:[23]

```
apt install kamailio kamailio-mysql-modules
```

5.1.2 Kamailio - konfigurace

Kamailio je úspěšně nainstalováno. Nyní je potřeba provést jeho základní konfiguraci. V prvním kroku jsem provedl nastavení spolupráce mezi Kamailiem a databází MariaDB. K tomu je určen soubor */etc/kamailio/kamctlrc*. Pro úpravu konfiguračního souboru jsem musel použít některý z dostupných textových editorů. V mém případě jsem zvolil editor Nano. V souboru jsem nejprve nastavil hodnotu položky *SIP_DOMAIN* tak, aby obsahovala IP adresu mého serveru. Druhou věcí bylo nastavení položky *DBENGINE*, kterou jsem nastavil na MYSQL. Na závěr jsem odkomentoval položku obsahující cestu k souboru *kamailio.pid*. Tento soubor obsahuje číslo procesu, které je Kamailiu přiřazeno při jeho spuštění. Položky upravené v tomto souboru vypadají následovně:

```
SIP_DOMAIN=158.196.244.195
DBENGINE=MYSQL
PID_FILE=/var/run/kamailio/kamailio.pid
```

Abychom umožnili Kamailiu komunikovat s databázovým serverem, nainstalovali jsme v dřívějším kroku MySQL modul, ten však musíme nejprve načíst. V hlavním konfiguračním souboru */etc/kamailio/kamailio.cfg* v sekci nastavení modulů jsem proto přidal následující položku:

```
loadmodule "db_mysql.so"
```

V sekci globálních parametrů jsem následně nastavil položku *listen*. Hodnota této položky nám určuje transportní protokol, který bude Kamailio vyžadovat pro SIP komunikaci a poté IP adresu a port, na kterém bude SIP server přijímat zprávy. Nastavená položka vypadá takto:

```
listen=udp:158.196.244.195:5060
```

Nyní můžeme vytvořit databázi, kterou bude Kamailio používat pro správu uživatelů. Vytvoření databáze jsem provedl následujícím příkazem:

```
kamdbctl create
```

Po zadání příkazu jsem byl vyzván k zadání hesla pro přístup k databázi MariaDB. Zadal jsem proto heslo, které jsem zvolil během instalace MariaDB. Vytvořená databáze nese název *kamailio*. K databázi mohou přistupovat dva uživatelé, *kamailio* a *kamailioro*. První z těchto uživatelů může z databáze číst a zároveň do ní zapisovat, zatímco uživatel *kamailioro* má právo pouze pro čtení.

Kamailio se stále nachází ve vypnutém stavu, abychom umožnili jeho spuštění, je zapotřebí odkomentovat v souboru */etc/default/kamailio* položku *RUN_KAMAILIO=yes*. Nyní je možné Kamailio spustit příkazem:[23]

```
systemctl start kamailio
```

Základní konfigurace SIP serveru Kamailio je úspěšně dokončena, lze proto otestovat jeho funkčnost. Pro tyto účely jsem pomocí níže uvedených příkazů přidal do databáze dva uživatele:

```
kamctl add alice test
kamctl add bob test
```

Položky následující za *kamctl add* jsou ve formátu "název uživatele" a "heslo uživatele". Přidání uživatelů vyžaduje zadání hesla. Pro uživatele *kamailio*, který má právo zápisu do databáze je heslo *kamailiorw*. Obsah databáze jsem vypsal příkazem:[23]

```
kamctl db show subscriber
```

id	username	domain	password
1	alice	158.196.244.195	test
2	bob	158.196.244.195	test

Obrázek 7: Výpis obsahu databáze

Mezi těmito uživateli bylo následně možné provést testovací hovor, jehož SIP komunikaci jsem odchytil pomocí terminálového nástroje *sngrep*. Na obrázku 8 lze vidět úspěšně sestavené a ukončené spojení mezi uživateli, jejichž zprávy byly předávány pomocí SIP Proxy Kamailio.

Obrázek 8: SIP komunikace s použitím SIP Proxy Kamailio

5.2 RTPProxy

5.2.1 RTPProxy - instalace

SIP Proxy Kamailio je připraveno, proto nyní můžeme rozšířit jeho funkčnost o jeden z dříve zmíněných modulů. Jako první jsem se rozhodl nainstalovat a nakonfigurovat modul RTPProxy. Pro instalaci jsem použil poslední dostupnou verzi z repozitářů. Konkrétně se jednalo o verzi 1.2.1. Stačilo zadat následující příkaz:

```
apt-get install rtpproxy
```

5.2.2 RTPProxy - konfigurace

První z věcí, kterou jsem vykonal po instalaci RTPProxy, bylo nastavení komunikačního soketu, na kterém bude komunikovat s Kamailiem. Existují dva typy soketu,: Unix soket a UDP soket. Já jsem v mé práci zvolil druhý z nich a jeho nastavení jsem provedl tímto příkazem:

```
rtpproxy -l 158.196.244.195 -s udp:127.0.0.1:7722 -u rtpproxy
```

Přepínač *-l* nastavuje IP adresu, na které je RTPProxy dostupné. Nastavení komunikačního soketu je zajištěno přepínačem *-s*. V našem případě se Kamailio i RTPProxy nachází na stejném zařízení, použil jsem proto IP adresu 127.0.0.1 (localhost), která umožňuje komunikaci mezi procesy pracujícími na stejném zařízení. Poslední přepínač *-u* definuje uživatele, pod kterým bude příkaz spuštěn. Druhou možností jak nastavit výše zmíněné parametry je použití konfiguračního souboru, který se nachází ve složce */etc/defult/rtpproxy*.[18]

5.2.3 RTPProxy - úprava Kamailia

Kamailio musíme o přítomnosti RTPProxy informovat. První věcí bylo načtení modulu *rtpproxy.so* v hlavním konfiguračním souboru Kamailia. V konfiguračním souboru se tento modul již nachází, je ovšem definován v bloku společně s modulem *nathelper.so*. Blok vypadá následovně:

```
#!ifdef WITH_NAT
loadmodule "nathelper.so"
loadmodule "rtpproxy.so"
#!endif
```

Pokud bychom chtěli tento blok načíst, stačilo by v úvodu konfiguračního souboru přidat položku *#!define WITH_NAT*. Pro naše účely ale nebylo potřeba služby *nathelperu* využívat. Vyjmul jsem proto z tohoto bloku položku *loadmodule "rtpproxy.so"* a přesunul ji mezi ostatní moduly, které jsou automaticky načteny při spuštění Kamailia.

Nyní musíme Kamailio informovat o komunikačním soketu, na kterém bude komunikovat s RTPProxy. Využijeme proto sekci nastavení parametrů modulů. Nastavení modulu RTPProxy se zde stejně jako v případě načítání modulu nachází v bloku *#!ifdef WITH_NAT*. Opět jsem tedy vyjmul položku týkající se RTPProxy a přesunul ji mimo tento blok. Samotné nastavení komunikačního soketu v Kamailiu vypadalo následovně:

```
modparam("rtpproxy", "rtpproxy.sock", "udp:127.0.0.1:7722")
```

Nastavení komunikace mezi Kamailiem a RTPProxy máme hotové. Cílem je ale zajistit, aby byly skrze RTPProxy směrovány veškeré RTP pakety. Je proto nutné upravit směrovací logiku Kamailia a použít jednu z funkcí, kterou modul RTPProxy nabízí. Informace o mediálním toku jsou přenášeny pomocí SDP protokolu, ten je posílán spolu se SIP zprávami INVITE a 200 OK. Musíme proto zajistit, aby byl obsah SDP přepsán a směroval tak veškerý tok na RTPProxy.

RTPProxy disponuje funkcí *rtpproxy_manage()*, která zajistí, že každá SIP zpráva obsahující SDP bude přepsaná do námi požadované podoby. Ve všech SIP zprávách obsahujících SDP potřebujeme přepsat prvky: *c*, *o* a číslo portu obsaženého v položce *m*, viz kapitola 2.3.1. Pro přepsání těchto parametrů disponuje modul RTPProxy takzvanými příznaky, které se chovají jako parametry funkce. Volání funkce bude v našem případě vypadat takto: *rtpproxy_manage("co")*. Funkci jsem poté zavolal v hlavním konfiguračním soboru Kamalia v sekci směrovacích bloků. Konkrétně v bloku *route[RELAY]*, který má na starost dodatečné úpravy SIP žádostí. Struktura bloku a volání funkce *rtpproxy_manage("co")* jsou viditelné níže.[24]

```
route[RELAY] {
      if (is_method("INVITE|BYE|SUBSCRIBE|UPDATE")) {
            if(!t_is_set("branch_route")) t_on_branch("MANAGE_BRANCH");
      }
      if (is_method("INVITE|SUBSCRIBE|UPDATE")) {
            if(!t_is_set("onreply_route")) t_on_reply("MANAGE_REPLY");
      }
      if (is_method("INVITE")) {
            if(!t_is_set("failure_route")) t_on_failure("MANAGE_FAILURE");
      }
      rtpproxy_manage("co");
      if (!t_relay()) {
            sl_reply_error();
      }
      exit;
}
```

Výpis 1: Volání funkce rtpproxy_manage() pro SIP žádosti obsahující SDP

Přepsání SIP žádostí, konkrétně zpráv INVITE obsahujících SDP, máme zajištěno. Stejnou činnost je potřeba provést i pro pozitivní finální odpovědi 200 OK. Pro řízení příchozích SIP odpovědí slouží blok *onreply_route[MANAGE_REPLY]*. Struktura bloku a volání funkce *rtpproxy_manage("co")* je zobrazena na níže uvedeném výpisu kódu.

```
onreply_route[MANAGE_REPLY] {
      xdbg("incoming reply\n");
      rtpproxy_manage("co");
      if(stauts=~"[12][0-9][0-9]") {
            route(NATMANAGE);
      }
}
```

Výpis 2: Volání funkce rtpproxy_manage() pro SIP odpovědi obsahující SDP

Úpravy hlavního konfiguračního souboru Kamailia vyjdou v platnost až po restartování Kamailia, které jsem provedl pomocí příkazu:

```
systemctl restart kamailio
```

Následně jsem provedl testovací hovor mezi Alicí a Bobem. Průběh hovoru jsem opět sledoval nástrojem *sngrep*. Veškerá pozornost byla směrována na SIP zprávy obsahující SDP. Na níže uvedených obrázcích lze vidět obsah SDP zprávy obsažené v žádosti INVITE, a to před/po průchodu SIP Proxy Kamailio.

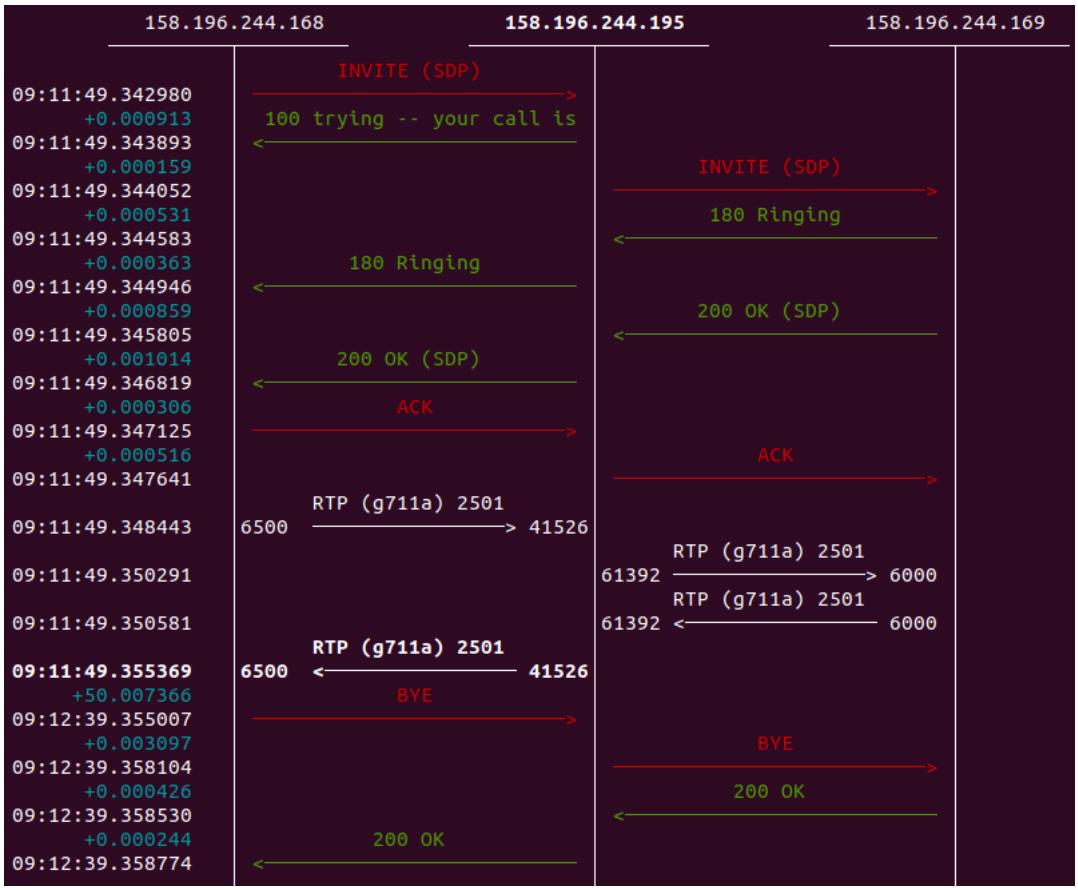
```
v=0
o=user1 53655765 2353687637 IN IP4 158.196.244.168
s=-
c=IN IP4 158.196.244.168
t=0 0
m=audio 6500 RTP/AVP 8 18
a=rtpmap:8 PCMA/8000
a=rtpmap:18 G729/8000
```

Obrázek 9: Obsah SDP před průchodem SIP Proxy Kamailio používající RTPProxy modul

```
v=0
o=user1 53655765 2353687637 IN IP4 158.196.244.195
s=-
c=IN IP4 158.196.244.195
t=0 0
m=audio 61392 RTP/AVP 8 18
a=rtpmap:8 PCMA/8000
a=rtpmap:18 G729/8000
a=nortpproxy:yes
```

Obrázek 10: Obsah SDP po průchodu SIP Proxy Kamailio používající RTPProxy modul

Průběh celé komunikace lze vidět na obrázku 11. V porovnání s obrázkem 8 je možné vidět, že jsou nyní veškerá mediální data směrována skrze RTPProxy.



Obrázek 11: Komunikace při použití SIP Proxy Kamailio a RTPProxy

5.3 RTPEngine

5.3.1 RTPEngine - instalace

Druhý modul, který jsem nainstaloval, byl RTPEngine. RTPEngine na rozdíl od RTPProxy nelze nainstalovat z repozitářů. Využil jsem proto oficiální stránky projektu, ze kterých jsem soubor stáhnul a následně zkompiloval. Stažení poslední dostupné verze 7.1.1.2 jsem provedl příkazem:

```
wget https://github.com/sipwise/rtpengine/archive/mr7.1.1.2.tar.gz
```

Obsah staženého souboru jsem extrahoval příkazem:

```
tar -xvzf mr7.1.1.2.tar.gz
```

Ve složce, ve které jsem se nacházel, se po zadání předchozího příkazu vytvořila stejnojmenná složka. Do této složky jsem se přesunul příkazem:

```
cd rtpengine-mr7.1.1.2
```

Sestavení balíčků, které jsou nezbytné pro následnou instalaci, jsem provedl příkazem:[19]

```
dpkg-buildpackage
```

Pokud systém obsahuje všechny potřebné doplňky, proběhne sestavení úspěšně. V mém případě výše uvedený příkaz vypsal řadu chybějících doplňků. Tyto doplňky jsem následně nainstaloval pomocí příkazu:

```
apt-get install "název_chybějícího_doplňku"
```

Jeden z chybějících doplňků ale nebylo možné tímto způsobem nainstalovat. Jednalo se o balíček *bcg729*. Využil jsem proto soubor nabízený přímo společností *Sipwise*, která stojí za projektem RTPEngine. Potřebné soubory jsem stáhnul a nainstaloval následovně:

1. `wget https://deb.sipwise.com/spce/mr6.2.1/pool/main/b/bcg729/libbcg729-0_1.0.4+git20180222-0.1~bpo9+1_amd64.deb`
2. `dpkg -i https://deb.sipwise.com/spce/mr6.2.1/pool/main/b/bcg729/libbcg729-0_1.0.4+git20180222-0.1~bpo9+1_amd64.deb`
3. `wget https://deb.sipwise.com/spce/mr6.2.1/pool/main/b/bcg729/libbcg729-dev_1.0.4+git20180222-0.1~bpo9+1_amd64.deb`
4. `dpkg -i https://deb.sipwise.com/spce/mr6.2.1/pool/main/b/bcg729/libbcg729-dev_1.0.4+git20180222-0.1~bpo9+1_amd64.deb`

Po instalaci všech potřebných doplňků jsem provedl opětovné zadání příkazu *dpkg-buildpackage*, tentokrát již proběhlo sestavení úspěšně a mimo složku *rtpengine-mr7.1.1.2* se vytvořila řada souborů s příponou *.deb*. Seznam vzniklých souborů byl následující:

- ngcp-rtpengine_7.1.1.2+0 mr7.1.1.2_all.deb
- ngcp-rtpengine-daemon_7.1.1.2+0 mr7.1.1.2_amd64.deb
- ngcp-rtpengine-iptables_7.1.1.2+0 mr7.1.1.2_amd64.deb
- ngcp-rtpengine-kernel-dkms_7.1.1.2+0 mr7.1.1.2_all.deb
- ngcp-rtpengine-kernel-source_7.1.1.2+0 mr7.1.1.2_all.deb
- ngcp-rtpengine-recording-daemon_7.1.1.2+0 mr7.1.1.2_amd64.deb
- ngcp-rtpengine-utils_7.1.1.2+0 mr7.1.1.2_all.deb

Výše uvedené soubory jsem nainstaloval příkazem:[19]

```
dpkg -i "název_souboru.deb"
```

5.3.2 RTPEngine - konfigurace

Software RTPEngine je nainstalován, nyní musíme provést jeho základní konfiguraci. Hlavní konfigurační soubor se nachází ve složce */etc/rtpengine* a nese název *rtpengine.sample.conf*. Tento název je nutné přejmenovat, protože při spuštění programu hledá *RTPEngine daemon* soubor *rtpengine.conf*. Výchozí soubor *rtpengine.sample.conf* je pouze ukázkový, ale pro naše účely disponuje všemi potřebnými parametry. Jeho přejmenování jsem provedl příkazem:

```
mv rtpengine.sample.conf rtpengine.conf
```

Přejmenovaný konfigurační soubor jsem následně otevřel textovým editorem *nano*. Stejně jako v případě RTPProxy bylo nutné nejprve specifikovat IP adresu, na které bude server dostupný. Pro zajištění komunikace mezi RTPEngine a SIP Proxy Kamailio jsem využil kontrolní protokol *ng*, který komunikoval na UDP soketu 127.0.0.1:8000. Následně jsem zvýšil rozsah portů, které může RTPEngine přidělovat. Tato hodnota nemá na komunikaci či výsledky závěrečných testů žádný vliv, může proto zůstat ve výchozí podobě. Nastavené položky a hodnoty vypadaly následovně:

```
interface = 158.196.244.195
listen-ng = 127.0.0.1:8000
port-min = 10000
port-max = 55000
```

Jak bylo zmíněno v kapitole 4.2, RTPEngine nabízí funkci *in-kernel packet forwarding*. Pro spuštění této funkce bylo nutné načíst *kernel modul* pomocí příkazu:

```
modprobe xt_RTPENGINE
```

Následně jsem přidat pravidlo do *iptables*, které zajistí přesměrování všech příchozích UDP (RTP) paketů do *kernel modulu*. Přidání pravidla do *iptables* vypadalo následovně:

```
iptables -I INPUT -p udp -j RTPENGINE --id 0
```

Přepínač *–id 0* představuje identifikační číslo směrovací tabulky. Tato hodnota je výchozí a odpovídá položce *table*, nacházející se v konfiguračním souboru */etc/rtpengine/rtpengine.conf*.

Základní konfigurace je tímto hotová, můžeme tedy celou službu spustit příkazem:[19]

```
service ngcp-rtpengine-daemon start
```

5.3.3 RTPEngine - úprava Kamailia

Základní úprava Kamailia pro spolupráci s RTPEngine se skládala z načtení modulu *rtpengine.so* a nastavení komunikačního soketu. Do hlavního konfiguračního souboru Kamailia, přesněji do sekce modulů, jsem proto přidal následující záznamy:

```
loadmodule "rtpengine.so"
modparam("rtpengine", "rtpengine_sock", "udp:127.0.0.1:8000")
```

Stejně jako v případě RTPProxy je nutné upravit směrovací logiku Kamailia. Pro přepsání SDP zpráv disponuje modul RTPEngine funkcí *rtpengine_manage()*. Opět bylo nezbytné použít takzvané příznaky, zajišťující přepsání již dříve zmíněných SDP položek. Na rozdíl od funkce *rtpproxy_manage("co")* nejsou příznaky specifikovány písmeny, ale jako slovní spojení oddělené mezerou. Volání funkce vypadalo následovně:

```
rtpengine_manage("replace-origin replace-session-connection")
```

Takto upravenou funkci jsem zavolal ve stejných blocích a na stejné pozici jako tomu bylo v případě RTPProxy, to znamená v bloku *route[RELAY]* a *onreply_reply[MANAGE_REPLY]*, viz kapitola 5.2.3. Následně jsem restartoval Kamailio příkazem:

```
systemctl restart kamailio
```

Následoval testovací hovor, ve kterém jsem primárně sledoval, zda dojde k přepisu položek obsažených v SDP. Výsledek je viditelný na níže uvedených obrázcích:

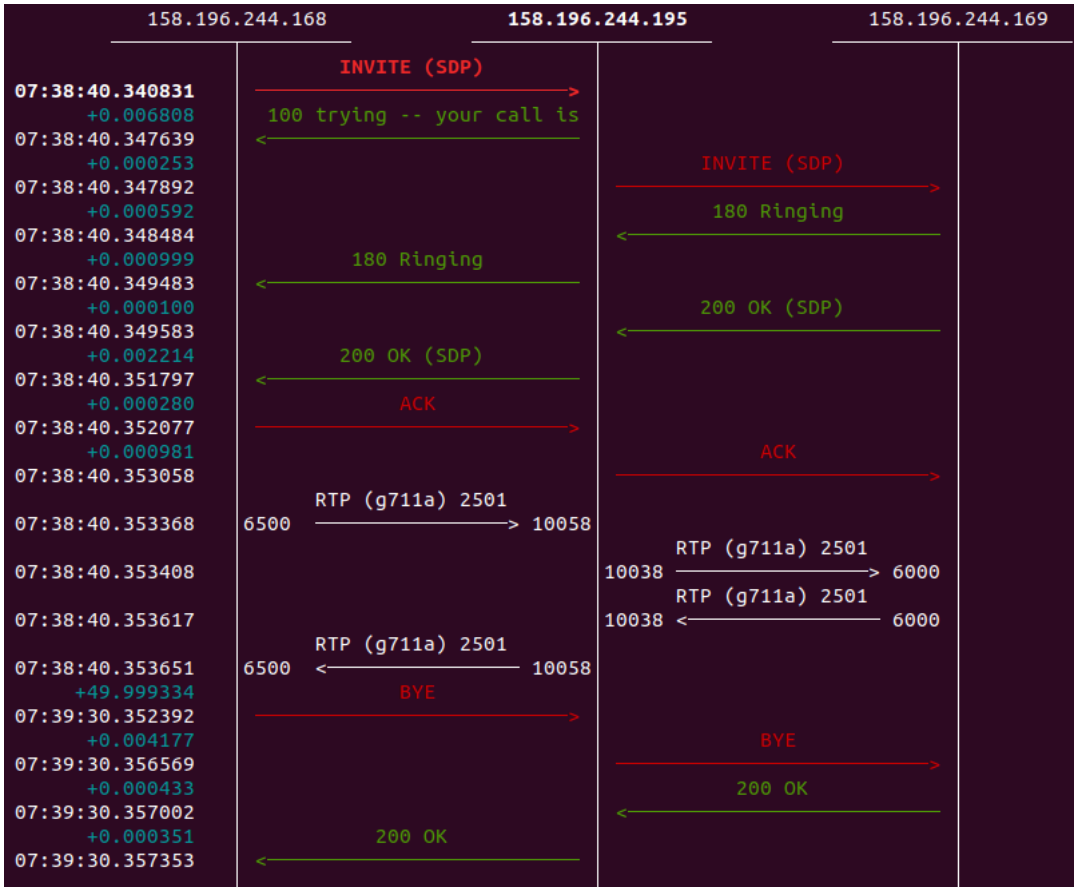
```
v=0
o=user1 53655765 2353687637 IN IP4 158.196.244.168
s=-
c=IN IP4 158.196.244.168
t=0 0
m=audio 6500 RTP/AVP 8 18
a=rtpmap:8 PCMA/8000
a=rtpmap:18 G729/8000
```

Obrázek 12: Obsah SDP před průchodem SIP Proxy Kamailio používající RTPEngine modul

```
v=0
o=user1 53655765 2353687637 IN IP4 158.196.244.195
s=-
c=IN IP4 158.196.244.195
t=0 0
m=audio 10038 RTP/AVP 8 18
a=rtpmap:8 PCMA/8000
a=rtpmap:18 G729/8000
a=sendrecv
a=rtcp:10039
a=ice-ufrag:w11IwdEE
a=ice-pwd:8la4aAa1cyhSDnXqkpV78ti7YL
a=candidate:mP9514SgvSCoSTtb 1 UDP 2130706431 158.196.244.195 10038 typ host
a=candidate:mP9514SgvSCoSTtb 2 UDP 2130706430 158.196.244.195 10039 typ host
```

Obrázek 13: Obsah SDP po průchodu SIP Proxy Kamailio používající RTPEngine modul

Celá komunikace odchycená pomocí nástroje *sngrep* vypadala takto:



Obrázek 14: Komunikace při použití SIP Proxy Kamailio a RTPEngine

5.4 MediaProxy

5.4.1 MediaProxy - instalace

Posledním z dostupných řešení určených pro předávání RTP toků byl modul MediaProxy. Záznam o umístění balíčku MediaProxy, který byl dostupný pro mou verzi operační systému Ubuntu Bionic Beaver jsem přidal do složky */etc/apt/source.list*. Přidané položky vypadaly následovně:[25]

```
deb http://ag-projects.com/ubuntu bionic main
deb-src http://ag-projects.com/ubuntu bionic main
```

Následně jsem stáhnul a přidal klíč příkazem:

```
wget -O /etc/apt/trusted.gpg.d/agp-debian-key.gpg http://download.ag-projects.
com/agp-debian-key.gpg
```

Níže uvedeným příkazem bylo nutné aktualizovat seznam balíčků.

```
apt-get update
```

Dále jsem se pokusil nainstalovat MediaProxy pomocí příkazu:

```
apt-get install mediaproxy-relay
```

Výše uvedený balíček se ale nepovedlo lokalizovat, a proto ho nebylo možné ani nainstalovat. Po konzultaci s vedoucím práce jsme zjistili, že pro mojí verzi operačního systému není výše uvedený balíček propagován a adresa s umístěním balíčku ho vůbec neobsahuje. Vedoucím mi bylo doporučeno vyzkoušet verzi pro operační systém Debian a konkrétně jeho vydání Strech. V souboru */etc/apt/source.list* jsem tedy smazal původně přidané záznamy a nahradil je těmito:

```
deb http://ag-projects.com/debian stretch main
deb-src http://ag-projects.com/debian stretch main
```

Stejně, jako v předchozím případě jsem aktualizoval seznam balíčků a následně se pokusil software nainstalovat znovu. To vše pomocí příkazů:

```
apt-get update
apt-get install mediaproxy-relay
```

Nyní již instalace proběhla úspěšně. Při zkušebním spuštění softwaru jsem zkontrolovat status služby příkazem:[25]

```
systemctl status mediaproxy
```

Výpis obsahoval řadu chybových hlášení, které byly spojeny s modulem systému. Vzniklé problémy jsem konzultoval s vedoucím práce a po vzájemné domluvě jsme došli k závěru, že s modulem MediaProxy nebudeme dále pracovat.

MediaProxy představuje nejstarší řešení pro předávání RTP toků, z námi zmíněných tří modulů, které jsou dostupné pro Kamailio. Z tohoto důvodu jsem od začátku práce očekával, že se bude jednat o nejméně výkonné řešení. Tento závěr jsem však nemohl prakticky ověřit, proto se zbytek práce zabývá pouze porovnáním modulů RTPProxy a RTPEngine.

6 Testování a vyhodnocení výsledků

Další úlohou práce bylo otestovat vliv jednotlivých modulů určených pro předávání audia/videa na zátěž serveru. Na virtuálním serveru, na kterém bylo nainstalováno Kamailio, byl tedy v jednu chvíli aktivní pouze jeden z modulů RTPProxy/RTPPengine. Následně jsem sledoval, jaký vliv má konkrétní modul na vytížení CPU (Central Processing Unit). V průběhu testování jsem se rozhodl pozorovat kromě vlivu jednotlivých modulů také vliv použitého kodeku. V mém případě jsem porovnával kodeky G.711 A-law a G.729, jejichž specifikace se nachází v tabulce 1. Abychom byli schopni vyvinout na testovaný server zátěž, bylo nezbytné najít způsob, jak generovat velké množství hovorů. Nástroj, který je pro tyto účely vhodný se nazývá SIPp.

SIPp je volně dostupný nástroj, určený pro testování SIP protokolu. Pro účely testování využívá dva základní SIP prvky, UAC a UAS. Chování těchto komponent je specifikováno pomocí takzvaných scénářů. Scénář představuje XML (eXtensible Markup Language) soubor, ve kterém jsou nadefinované jednotlivé SIP zprávy, které bude UA posílat/přijímat. Důležitou funkcí, kterou SIPp nabízí je posílání mediálních (RTP) dat. Tato funkce je pro nás velmi důležitá, jelikož potřebujeme otestovat moduly určené právě pro předávání RTP dat. Aby bylo možné simulovat hovory mezi UAC a UAS, byly mi vedoucím této práce vyhrazeny další dva virtuální servery, na kterých jsem SIPp nainstaloval v jeho poslední verzi 3.5.2. Před samotnou instalací bylo potřeba doinstalovat řadu doplňků, které jsou nezbytné pro správné fungování tohoto testovacího nástroje. Seznam a postup instalace jednotlivých doplňků byl následující:[26][28]

```
apt-get install build-essential
apt-get install openssl libssl1.0-dev
apt-get install libpcap-dev libnet-pcap-perl
apt-get install libncurses5-dev libncursesw5-dev
```

Následně jsem SIPp stáhnul a rozbalil pomocí příkazů:[28]:

```
wget https://github.com/SIPp/sipp/releases/download/v3.5.2/sipp-3.5.2.tar.gz
tar -xvzf sipp-3.5.2.tar.gz
```

Do rozbaleného adresáře jsem se přemístil příkazem:

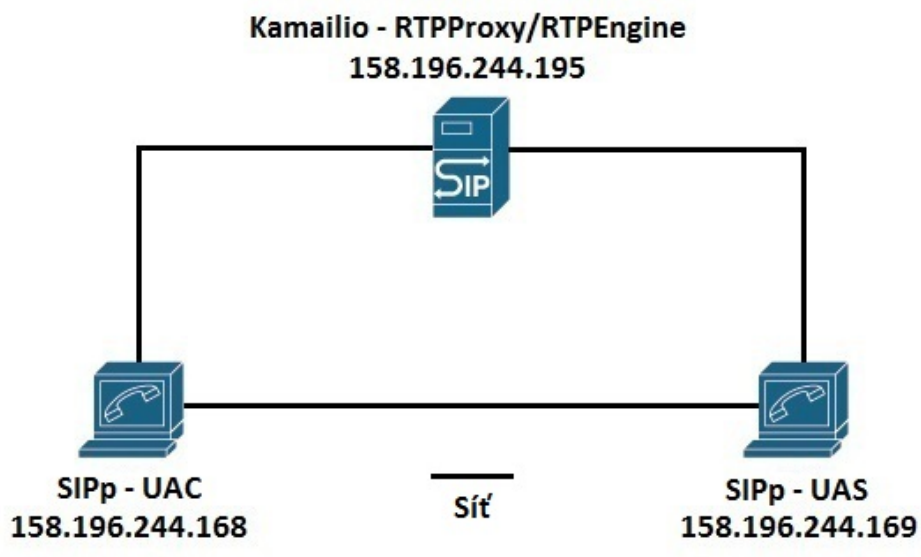
```
cd sipp-3.5.2
```

Zadáním níže uvedeného příkazu proběhla kontrola, ve které bylo ověřeno, zda jsou nainstalovány všechny potřebné doplňky.

```
./configure --with-pcap
```

Poté už bylo možné provést samotné sestavení programu SIPp příkazem:[28]

```
make
```



Obrázek 15: Testovací topologie[21]

Na obrázku číslo 15 lze vidět testovací topologii složenou ze tří virtuálních serverů. Každý z těchto serverů disponoval ve výchozím stavu dvoujádrovým procesorem Intel Xeon 2,1 GHz, 4 GB RAM a operačním systémem Ubuntu 18.04 LTS - 64bit.

Na serverech, na kterých byl nainstalován SIPp bylo nejprve nutné vytvořit scénáře definující chování během SIP dialogu. SIPp pro tyto účely nabízí na svých stránkách řadu volně dostupných scénářů, které byly pro testování dostačující. V mém případě jsem použil scénáře *uac.xml* a *uas.xml*.[27]

Scénáře před prvním použitím vyžadovaly řadu úprav. Jednou z klíčových úprav bylo přizpůsobit scénáře Kamailiu, které se udržuje v signalizaci až do skončení hovoru pomocí funkce record_route(). Upravil jsem proto jednotlivé SIP zprávy obsažené ve scénářích tak, aby s touto funkcí spolupracovaly. V *uac* scénáři jsem do očekávaných odpovědí na žádost INVITE přidal parametr *rrs* (record route set) a nastavil ho na hodnotu *true*. Tento parametr zajistí uložení záhlaví položky *Record-route* z přijaté SIP odpovědi. Uloženou hodnotu později použijeme pro odesílání dalších SIP zpráv, a to zavoláním klíčového slova *[routes]*. Struktura očekávaných SIP zpráv s přidaným parametrem *rrs* je následující:

```
<recv response="180" rrs="true" optional="true"></recv>
<recv response="200" rrs="true" rtd="true" crlf="true"></recv>
```

Tag *recv* udává typ SIP zprávy, který se očekává. Konkrétní označení očekávané zprávy je poté uvedeno v parametru *response*. SIP zpráva 180 navíc obsahuje parametr *optional*, který udává, že se jedná o volitelnou/nepovinnou SIP zprávu.

Do SIP žádosti ACK a BYE jsem poté přidal položku *[routes]*, jejíž hodnota bude v našem případě obsahovat adresu Kamailia, které se chce udržet v signalizaci. Záhlaví těchto žádostí s přidanou položkou *[routes]* vypadá následovně:

```
ACK sip:[service]@[remote_ip]:[remote_port] SIP/2.0 [routes]
BYE sip:[service]@[remote_ip]:[remote_port] SIP/2.0 [routes]
```

Poslední nezbytné úpravy týkající se *uac* scénáře zahrnovaly přidání položek do SDP zprávy, obsažené v žádosti INVITE a definování bloku zajišťujícího spuštění nahrávky simulující hovor. V SDP zprávě bylo nutné specifikovat použité kodeky, v našem případě se jednalo o kodeky G.711 A-law a G.729. Přidané záznamy vypadaly následovně:

```
m=audio [auto_media_port] RTP/AVP 8 18
a=rtpmap:8 PCMA/8000
a=rtpmap:18 G729/8000
```

Číselné označení 8 a 18 definuje typ užitečné zátěže, takzvaný *Payload type*. Tuto hodnotu má každý kodek jinou. Nyní už stačilo pouze nadefinovat blok, ve kterém se provede spuštění nahrávky simulující hovor. Pro každý kodek jsem si vytvořil vlastní nahrávku o délce 50 sekund, a to pomocí nástroje Wireshark. Wireshark totiž umožnuje uložit odchycený hovor do formátu *.pcap*, který je nástrojem SIPp podporován. Samotný blok jsem vložil za zprávu ACK, která potvrzuje sestavení spojení. Blok obsahující spuštění nahrávky je viditelný níže.[27]

```
<nop>
  <action>
    <exec play_pcap_audio="/home/vanek/sipp-3.5.2/pcap/g711a.pcap"/>
  </action>
</nop>
<pause miliseconds="50000">
```

Výpis 3: Blok definující posílání médií v SIPp scénáři

Tímto byl *uac* scénář připraven k použití. Následně bylo nutné upravit i scénář *uas*. Stejně jako v případě *uac* bylo i zde zapotřebí přidat parametr *rrs*, a to konkrétně do zprávy INVITE, kterou *uas* očekává.

```
<recv request="INVITE" rrs="true" crlf="true"></recv>
```

Obsah položky *Record-route*, který *uas* obdržel v žádosti INVITE si uloží a poté ho ve zprávách 180 Ringing a 200 OK posílá na *uac* pomocí položky *[last_Record-Route]*. Volání této položky ve zprávě 180 Ringing vypadá následovně:[27]

```
<send>
  <![CDATA[
    SIP/2.0 180 Ringing
    [last_Record-Route:]
    [last_Via:]
    [last_From:]
    [last_To:];tag=[pid]SIPpRag01[call_number]
    [last_Call-ID:]
    [last_CSeq:]
    Contact: <sip:[local_ip]:[local_port];transport=[transport]>
    Content-Length> 0
  ]]>
</send>
```

Výpis 4: Přidání položky last_Record-Route do zprávy 180 Ringing v SIPp scénáři

Poslední nutnou úpravou v tom scénáři bylo specifikování použitých kodeků v SDP, která se v případě *uas* nachází ve zprávě 200 OK. Přidané položky jsou totožné s těmi, které jsme dříve přidali do scénáře *uac*, viz výpis uvedený níže:

```
m=audio [auto_media_port] RTP/AVP 8 18
a=rtpmap:8 PCMA/8000
a=rtpmap:18 G729/8000
```

V tuto chvíli již scénáře s Kamailiem plně spolupracovaly, nebylo však možné sestavit větší množství souběžných hovorů. Příčinou byl výchozí limit operačního systému, který definuje maximální počet otevřených souborů. Výchozí hodnota je 1024, pro účely následného testování jsem na všech třech virtuálních serverech hodnotu navýšil na 150000 pomocí příkazu:

```
ulimit -n 150000
```

Vedoucím práce mi bylo doporučeno zvýšit i hodnotu *stacku*, do kterého se ukládají například hodnoty proměnných. Výchozí hodnotu 8192 kB jsem změnil na neomezenou příkazem:

```
ulimit -s unlimited
```

Po těchto úpravách jsem provedl řadu zkušebních testů, ale v žádném z nich jsem se nebyl schopen přiblížit maximálnímu vytížení serveru, na kterém Kamailio pracovalo. Problém způsoboval SIPp UAC, který po chvíli testování zahltil CPU do takové míry, že již nebylo možné počet souběžných hovorů zvyšovat. Po domluvě s vedoucím práce byl na tomto serveru navýšen počet jader CPU z původních 2 na 8. Zároveň jsme provedli redukci jader na serveru s Kamailiem, a to ze 2 na 1. V dalších testech již bylo možné vytížit CPU Kamailia až do jeho hranice, ale průběhy testů vykazovaly velké množství retransmisí. Za pomoci nástroje *tcpdump* jsem se proto rozhodl odchytit provoz jdoucí skrze Kamailio, aby bylo následně možné provést jeho analýzu. Odchycení provozu jsem provedl tímto příkazem:

```
tcpdump -w 1.pcap -i ens160
```

Odchycená data jsem analyzoval v programu *Wireshark*. Výsledky ukázaly přítomnost velkého množství ICMP (Internet Control Message Protocol) paketů, které byly generovány ze strany SIPp UAC. Po důkladnější analýze jsem zjistil, že SIPp UAC nenaslouchal příchozím RTP paketům a příchozí data tedy neměl kdo zpracovávat. Z tohoto důvodu musel kernel generovat ICMP pakety, jejichž množství postupně zcela zahltilo CPU. Řešením bylo zakázat příchozí komunikaci (RTP pakety) na porty v rozsahu 6000 - 65535 pomocí tohoto příkazu:

```
iptables -A INPUT -p udp --dport 6000:65535 -j DROP
```

Přidáním tohoto pravidla do firewallu se zcela změnilo chování a průběh testování. Během následujících testů již nevznikaly téměř žádné retransmise a Kamailio bylo možné vytížit pouze jedním UAC/UAS pracujícím na jednom jádru. V následném testování jsem již s počty jader CPU nemanipuloval, Kamailio tedy disponovalo jednojádrovým procesorem a 4GB RAM.

6.1 Průběh a analýza testů

Na začátku testování konkrétního modulu a kodeku jsem nejprve zjistil počet souběžných hovorů, při kterém se vytížení CPU blížilo jeho maximu. Záměrně jsem se vyhýbal hodnotě vytížení 100%, protože by poté došlo k přetížení CPU a následnému selhávání hovorů. Úroveň vytížení procesoru jsem proto sledoval nástrojem *sysstat*, který mi na konci každého testu vypsal průměrnou hodnotu vytížení.

Zjištěný počet souběžných hovorů, který jsem považoval za maximální, byl následně rozdělen do n měření s krokem 250 souběžných hovorů. Pro každý z těchto kroků jsem provedl 5 nezávislých testů o délce 180 sekund. Z výsledků jsem vypočítal průměrné vytížení CPU a také hodnotu směrodatné odchylky. Po každém z testů jsem provedl restart Kamailia a aktuálně používaného RTP proxy serveru. Pokud jsem například dosáhl maximálního počtu hovorů 1450, tak jsem tuto hodnotu rozdělil do měření s počtem souběžných hovorů 0, 250, 500, 750, 1000, 1250 a 1450, přičemž velikost posledního kroku jsem přizpůsobil zjištěnému maximu.

Před každým testem bylo nutné spustit nejprve SIPp UAS, protože kdybychom spustili nejdříve SIPp UAC, neměl by kdo zpracovávat hovory, které byly vygenerovány ještě před spuštěním UAS. Příkaz, kterým jsem spustil SIPp UAS byl následující:

```
taskset -c 0 ./sipp -sf uas.xml -i 158.196.244.169 -p 5061 -rtp_echo
```

Příkaz *taskset -c 0* není spojen se SIPp, ale určuje pouze to, že bude SIPp UAS pracovat na jádru s pořadovým číslem 1(logické číslo 0). Použití této položky bylo spojeno pouze se sledováním vlivu UAC/UAS na vytížení CPU, nemělo však žádný vliv na výsledku testů. Přepínač *-sf* určuje použití vlastního scénáře, přepínač *-i* nastavuje IP adresu a port, kterou UAS použije pro pole *Contact*, *Via* a *from* v SIP hlavičce. Přepínač *-rpt_echo* zajistí, že všechny RTP pakety, které UAS příjme budou poslány zpět odesílateli. Tím zajistíme simulaci obousměrného hovoru. Následně bylo možné spustit SIPp UAC příkazem:

```
taskset -c 0 ./sipp -sf uac.xml 158.196.244.169:5061 -i 158.196.244.168 -p 5061
-rsa 158.196.244.195:5060 -skip_rlimit -mp 6500 -r 29
```

První adresa určuje IP adresu a port volaného, v našem případě se jedná o SIPp UAS. Přepínač *-rsa* určuje IP adresu a port vzdáleného serveru, který bude předávat SIP zprávy, jedná se proto o adresu Kamailia. Přepínač *-skip_rlimit* umožní překročit limit pro maximální počet otevřených souborů, jehož výchozí hodnota je 1024. Přepínač *-mp* nastaví číslo portu, na kterém bude UAC přijímat RTP pakety. V mém případě jsem záměrně použil číslo z rozsahu portů, pro které jsem dříve zakázal příchozí komunikaci. Přepínač *-r* určuje počet vytvořených hovorů za sekundu. Pokud tuto hodnotu vynásobíme délkou našich nahrávek (50 sekund), dostaneme se na námi požadovaný počet souběžných hovorů. S tímto přepínačem ale nebylo možné postupovat s krokem menším než 50 souběžných hovorů. V případě, kdy bylo nutné zajistit jemnější doladění, jsem využil přepínač *-rp*. Ten definuje periodu, s jakou mají být hovory generovány. Například pokud bych chtěl dosáhnout 1525 souběžných hovorů, vypadalo by nastavení přepínačů následovně: *-r* 61 *-rp* 2000, přičemž hodnota přepínače *-rp* je udávaná v milisekundách. Poté, co byly oba SIPp prvky spuštěny, jsem čekal přibližně jednu minutu, než se počet souběžných hovorů ustálil na požadované hodnotě. Následně jsem na serveru s Kamailiem spustil měření sledující vytížení CPU. Příkaz pro spuštění nástroje *sysstat* vypadal následovně:[27]

```
sar -u 1 180
```

Přepínač *-u* určuje, že do výsledné hodnoty vytížení budou zahrnuty všechny procesory. V našem případě pouze jeden. Další hodnoty specifikují, jak často se měření provede a celkový počet měření. Během našeho testování se měření provede jednou za sekundu a bude jich celkem 180. Celé měření proto bude trvat přesně 3 minuty.

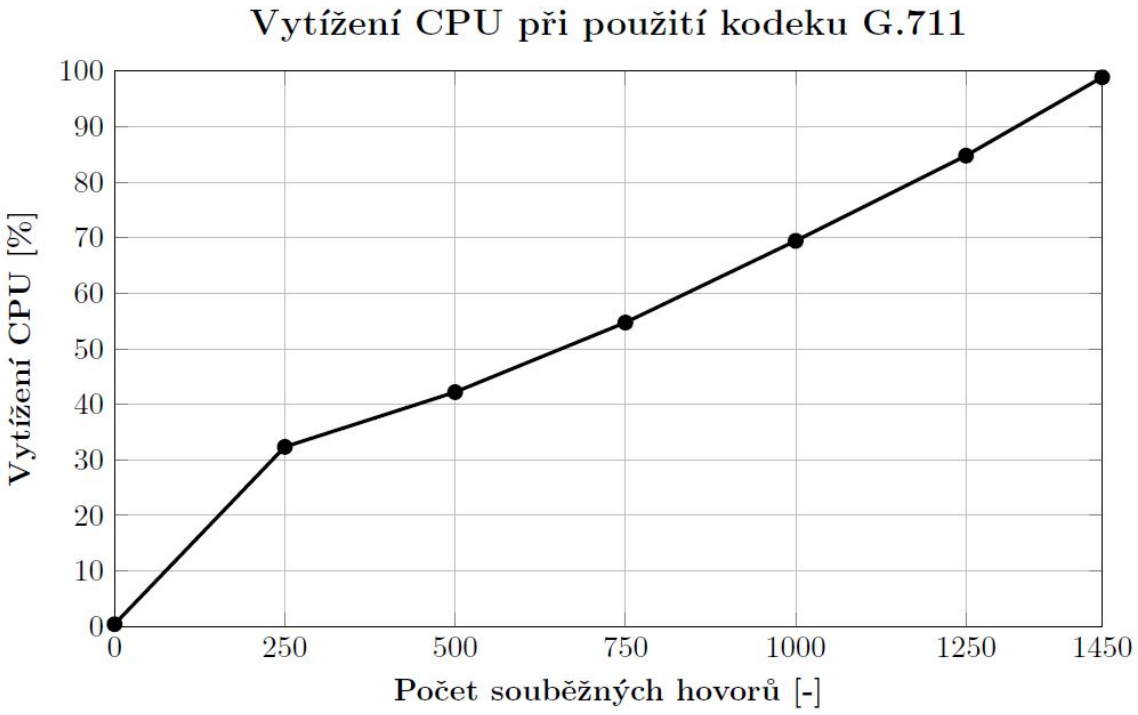
Ve zbylé části kapitoly jsou uvedeny výsledky testování pro jednotlivé moduly a kodeky. Průběh testování byl obdobný jako ve výše uvedeném příkladu. Jediná změna, která probíhala v rámci jednotlivých testů/měření, byla v přepínači *-r*, případně *-rp*.

6.2 Výsledky testování - RTPProxy

6.2.1 RTPProxy - kodek G.711

	Počet souběžných hovorů						
	0	**250**	**500**	**750**	**1000**	**1250**	**1450**
Vytížení CPU - test č. 1	0,42	31,95	42,36	55,07	68,66	84,63	98,38
Vytížení CPU - test č. 2	0,41	32,66	42,71	55,04	69,02	85,20	98,93
Vytížení CPU - test č. 3	0,43	32,31	42,05	54,42	69,71	85,22	99,05
Vytížení CPU - test č. 4	0,44	32,00	42,47	54,21	69,79	84,51	99,04
Vytížení CPU - test č. 5	0,39	32,83	41,46	54,93	70,05	84,36	99,08
Vytížení CPU - průměr	0,42	32,35	42,21	54,73	69,45	84,78	98,90
Směrodatná odchylka	0,02	0,35	0,43	0,35	0,52	0,36	0,26

Tabulka 2: Naměřené hodnoty - RTPProxy (G.711)

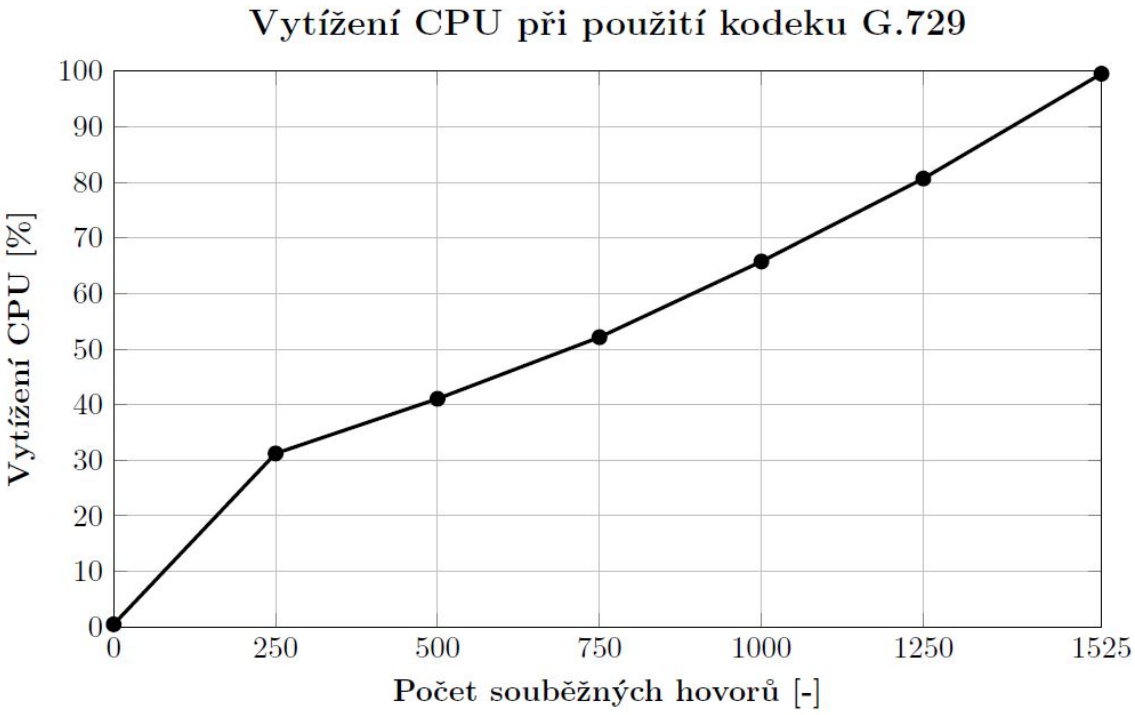


Obrázek 16: Výsledek testování - RTPProxy (G.711)

6.2.2 RTPProxy - kodek G.729

	Počet souběžných hovorů						
	0	**250**	**500**	**750**	**1000**	**1250**	**1525**
Vytížení CPU - test č. 1	0,49	31,49	40,97	52,01	65,45	80,63	99,48
Vytížení CPU - test č. 2	0,50	31,43	41,20	52,49	65,69	80,57	99,55
Vytížení CPU - test č. 3	0,49	31,06	40,59	52,12	65,86	80,91	99,38
Vytížení CPU - test č. 4	0,51	30,87	41,47	51,98	65,76	80,41	99,77
Vytížení CPU - test č. 5	0,47	31,30	41,10	52,11	66,09	80,90	99,57
Vytížení CPU - průměr	0,49	31,23	41,07	52,14	65,77	80,68	99,55
Směrodatná odchylka	0,01	0,23	0,29	0,18	0,21	0,19	0,13

Tabulka 3: Naměřené hodnoty - RTPProxy (G.729)



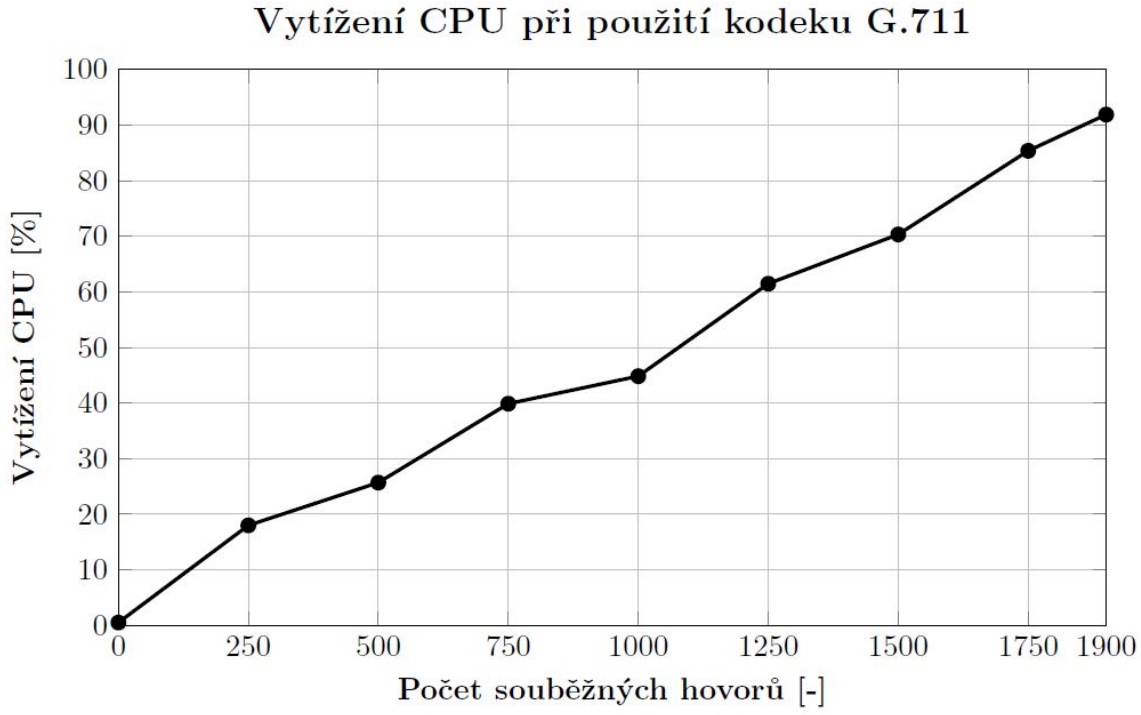
Obrázek 17: Výsledek testování - RTPProxy (G.729)

6.3 Výsledek testování - RTPEngine bez funkce In-kernel packet forwarding

6.3.1 RTPEngine bez funkce In-kernel packet forwarding - kodek G.711

	Počet souběžných hovorů								
	0	**250**	**500**	**750**	**1000**	**1250**	**1500**	**1750**	**1900**
Vytížení CPU - test č. 1	0,51	15,90	26,97	40,22	46,76	61,57	67,45	84,96	91,13
Vytížení CPU - test č. 2	0,54	19,24	25,16	42,00	43,51	62,34	69,17	85,08	92,01
Vytížení CPU - test č. 3	0,51	16,81	25,60	43,38	45,29	63,50	69,71	85,90	91,44
Vytížení CPU - test č. 4	0,49	19,06	25,03	35,21	44,59	59,48	71,87	85,67	92,63
Vytížení CPU - test č. 5	0,54	19,00	25,73	38,64	44,04	60,48	73,47	85,29	92,27
Vytížení CPU - průměr	0,52	18,00	25,70	39,89	44,84	61,47	70,33	85,38	91,90
Směrodatná odchylka	0,02	1,38	0,69	2,84	1,13	1,40	2,11	0,36	0,55

Tabulka 4: Naměřené hodnoty - RTPEngine bez funkce In-kernel packet forwarding (G.711)

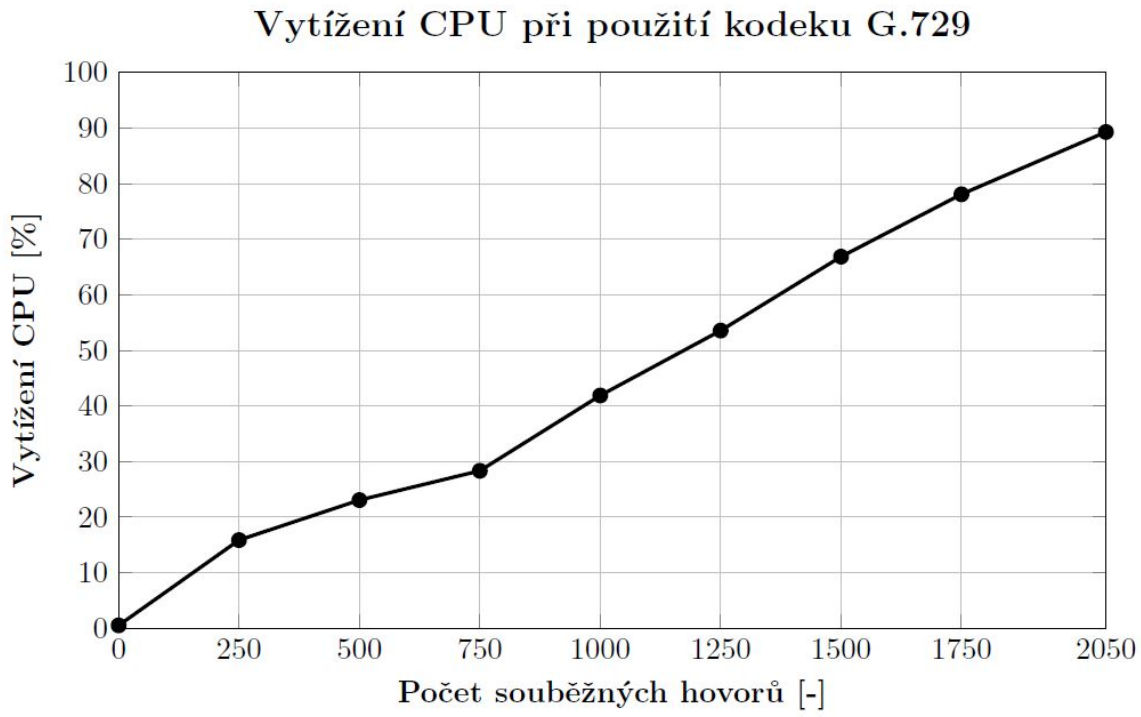


Obrázek 18: Výsledek testování - RTPEngine bez funkce In-kernel packet forwarding (G.711)

6.3.2 RTPEngine bez funkce In-kernel packet forwarding - kodek G.729

	Počet souběžných hovorů								
	0	**250**	**500**	**750**	**1000**	**1250**	**1500**	**1750**	**2050**
Vytížení CPU - test č. 1	0,55	14,71	23,74	31,22	41,62	54,20	65,38	79,46	89,47
Vytížení CPU - test č. 2	0,50	15,48	22,80	29,82	42,02	54,02	67,19	76,52	89,02
Vytížení CPU - test č. 3	0,51	15,02	22,54	27,90	41,93	53,98	67,19	77,27	89,77
Vytížení CPU - test č. 4	0,56	16,84	22,60	27,14	41,71	52,28	68,70	79,23	89,24
Vytížení CPU - test č. 5	0,51	17,32	23,68	25,63	42,18	53,35	65,86	77,97	88,99
Vytížení CPU - průměr	0,53	15,87	23,07	28,34	41,89	53,57	66,86	78,09	89,30
Směrodatná odchylka	0,02	1,03	0,53	1,97	0,20	0,70	1,17	1,13	0,29

Tabulka 5: Naměřené hodnoty - RTPEngine bez funkce In-kernel packet forwarding (G.729)



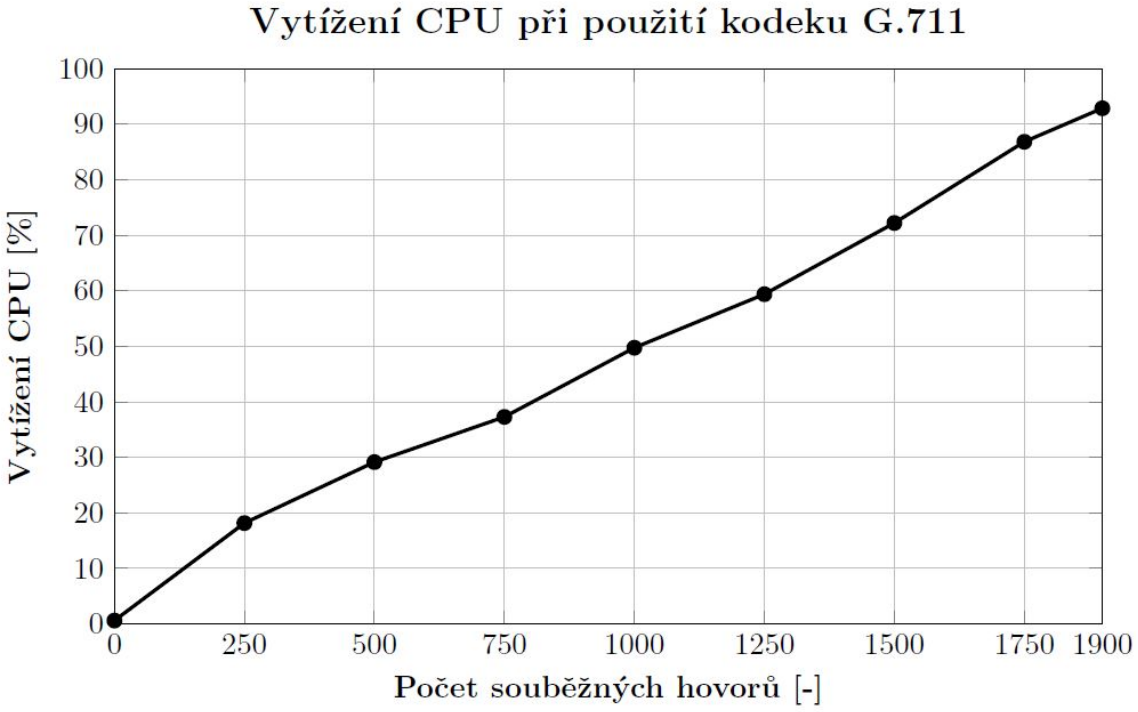
Obrázek 19: Výsledek testování - RTPEngine bez funkce In-kernel packet forwarding (G.729)

6.4 Výsledek testování - RTPEngine s funkcí In-kernel packet forwarding

6.4.1 RTPEngine s funkcí In-kernel packet forwarding - kodek G.711

	Počet souběžných hovorů								
	0	**250**	**500**	**750**	**1000**	**1250**	**1500**	**1750**	**1900**
Vytížení CPU - test č. 1	0,60	20,12	30,71	37,52	49,09	57,89	72,62	86,71	92,70
Vytížení CPU - test č. 2	0,61	18,42	30,52	39,93	49,40	58,04	74,56	87,21	93,36
Vytížení CPU - test č. 3	0,61	17,53	27,78	38,09	50,13	59,84	71,46	87,13	92,77
Vytížení CPU - test č. 4	0,59	17,33	29,43	35,10	50,20	60,78	71,99	85,83	92,86
Vytížení CPU - test č. 5	0,62	17,41	27,29	35,78	49,86	60,35	70,40	87,27	93,14
Vytížení CPU - průměr	0,61	18,16	29,15	37,28	49,74	59,38	72,21	86,83	92,97
Směrodatná odchylka	0,01	1,05	1,40	1,72	0,43	1,19	1,38	0,54	0,25

Tabulka 6: Naměřené hodnoty - RTPEngine s funkcí In-kernel packet forwarding (G.711)

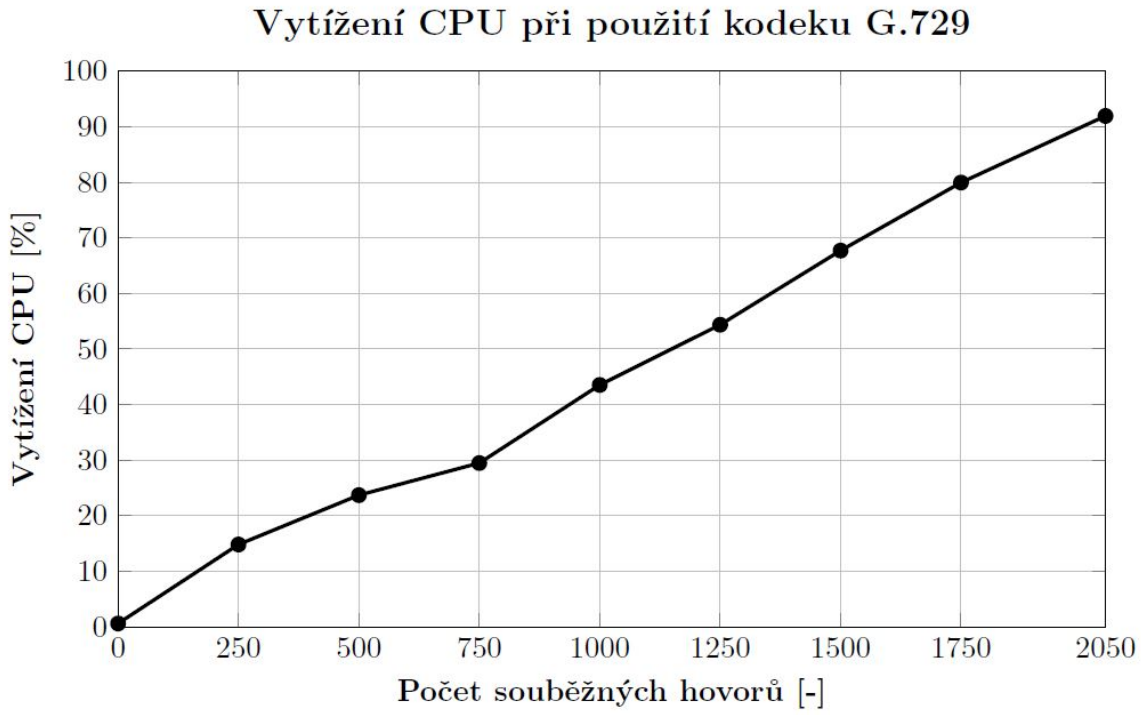


Obrázek 20: Výsledek testování - RTPEngine s funkcí In-kernel packet forwarding (G.711)

6.4.2 RTPEngine s funkcí In-kernel packet forwarding - kodek G.729

	Počet souběžných hovorů								
	0	**250**	**500**	**750**	**1000**	**1250**	**1500**	**1750**	**2050**
Vytížení CPU - test č. 1	0,56	14,86	24,55	27,67	42,77	54,45	67,05	79,04	92,48
Vytížení CPU - test č. 2	0,63	13,99	23,12	28,68	43,57	52,57	68,75	79,96	92,54
Vytížení CPU - test č. 3	0,59	14,93	24,36	29,28	44,68	54,07	67,37	80,70	92,54
Vytížení CPU - test č. 4	0,62	15,41	23,54	30,97	43,91	54,51	68,35	80,21	91,45
Vytížení CPU - test č. 5	0,68	14,95	22,96	30,82	42,73	56,22	67,00	79,83	90,64
Vytížení CPU - průměr	0,62	14,83	23,70	29,48	43,53	54,36	67,70	79,95	91,93
Směrodatná odchylka	0,04	0,46	0,64	1,26	0,73	1,16	0,71	0,54	0,77

Tabulka 7: Naměřené hodnoty - RTPEngine s funkcí In-kernel packet forwarding (G.729)



Obrázek 21: Výsledek testování - RTPEngine s funkcí In-kernel packet forwarding (G.729)

7 Závěr

Tato bakalářská práce se zabývá technologií VoIP. Konkrétním cílem práce bylo otestovat a porovnat dostupná řešení pro předávání RTP toků, která jsou určená pro spolupráci se SIP serverem Kamailio.

V teoretické části této práce jsme se seznámili se základními vlastnostmi technologie VoIP. Následně jsme podrobně popsali jednotlivé protokoly, které se s touto technologií pojí a které charakterizují její vlastnosti. Z těchto protokolů byl kladen důraz zejména na signalizační protokol SIP. V další části jsme se zabývali popisem SIP serveru Kamailio a zejména strukturou jeho architektury. Dále byly zmíněny obecné vlastnosti RTP proxy serverů, jejich výhody/nevýhody a také dostupná řešení RTP proxy serverů, která jsou určená pro SIP server Kamailio.

Hlavním bodem bakalářské práce bylo nainstalovat a nakonfigurovat SIP Proxy Kamailio do takové podoby, aby plně spolupracovalo s dostupnými řešeními pro směrování RTP toků. Do SIP proxy Kamailio se povedlo naintegrovat modul RTPProxy a RTPEngine. Poslední řešení v podobě modulu MediaProxy se nepovedlo zprovoznit z důvodu chybějícího instalačního balíčku, který pro námi používanou verzi operačního systému nebyl dostupný. V závěrečné pasáži praktické části jsme provedli testovaní modulů RTPProxy a RTPEngine. Pro účely testování jsme použili nástroj SIPp, jehož scénáře jsme upravili do takové podoby, aby jimi bylo možné plně otestovat jednotlivá řešení pro předávání RTP toků. V průběhu testování jsme sledovali vliv použitého modulu a kodeku na vytížení CPU serveru.

S modulem RTPProxy jsme se dostali na hranici 1525 souběžných hovorů pro kodek G.729 a na hranici 1450 souběžných hovorů při použití kodeku G.711. Z výsledků testů se však jako nejvýkonnější dostupné řešení pro předávání RTP toků ukázal RTPEngine. S tímto modulem jsme dosáhli hranice 2050 souběžných hovorů pro kodek G.729 a 1900 souběžných hovorů pro kodek G.711, což představuje nárust přibližně o 32,5% v porovnání s modulem RTPProxy. V případě modulu RTPEngine jsme rovněž sledovali vliv funkce In-kernel packet forwarding. Z výsledků je patrné, že provozování tohoto modulu s vypnutou funkcí In-kernel packet forwarding bylo optimalizováno natolik dobře, že jsou výsledky téměř shodné s testováním, ve kterém je tato funkce aktivní. Průběhy testů byly při použití modulu RTPProxy velmi stabilní, bylo proto možné se velmi blízce přiblížit hranici 100% vytížení CPU, zatímco testy modulu RTPEngine vykazovaly kolísání ve vytížení CPU v rozmezí +- 8%. To je hlavní důvod, proč se nebylo možné přiblížit hranici 100% jako v případě modulu RTPProxy. V rámci testů jednotlivých modulů bylo vždy dosaženo maximální hodnoty souběžných hovorů při použití kodeku G.729.

Přínosem této bakalářské práce je přehled vlivu jednotlivých modulů a kodeků na vytížení serveru. Výsledky mohou být užitečné při výběru konkrétního řešení pro předávání RTP toků a také při výběru kodeku. Zároveň může práce sloužit jako návod na instalaci a základní konfiguraci SIP Proxy Kamailia či modulů RTPEngine a RTPProxy.

Literatura

[1] *Jak funguje VOIP?* [online]. [cit. 2019-03-02]. Dostupné z: http://www.earchiv.cz/b06/b0401003.php3

[2] WALLINGFORD, Ted. *Switching to VoIP.* Sebastopol, CA: O'Reilly, 2005. ISBN 978-0-596-00868-0.

[3] HARTPENCE, Bruce. *Packet guide to voice over IP.* Tokyo: O'Reilly, 2013. ISBN 978-1-449-33967-8.

[4] *VoIP and the OSI model* [online]. [cit. 2019-03-02]. Dostupné z: http://voip-hosting-providers.blogspot.com/2015/04/voip-and-osi-model.html

[5] HARTPENCE, Bruce. *Packet Guide to Core Network Protocols.* CA: O'Reilly, 2011. ISBN 978-1-449-30653-3.

[6] JOHNSTON, Alan B. *SIP: understanding the session initiation protocol.* 3rd ed. Boston: Artech House, 2009. ISBN 978-1-60783-995-8 .

[7] *RFC 791 - Internet Protocol* [online]. [cit. 2019-03-29]. Dostupné z: https://tools.ietf.org/html/rfc791

[8] *TCP a UDP* [online]. [cit. 2019-03-09]. Dostupné z: http://www.earchiv.cz/anovinky/ai1864.php3

[9] *Transmission Control Protocol (TCP)* [online]. [cit. 2014-03-09]. Dostupné z: http://www.cs.vsb.cz/grygarek/SPS/lect/TCP/tcp.html

[10] VOZŇÁK, Miroslav. *Technologie a protokoly multimediálních komunikací pro integrovanou výuku VUT a VŠB-TUO.* Ostrava: Vysoká škola báňská - Technická univerzita Ostrava, 2014. ISBN 978-80-248-3326-2.

[11] *SIP Dialog and Routing of requests* [online]. [cit. 2019-03-30]. Dostupné z: https://www.tech-invite.com/fo-sip/tinv-fo-sip-dialog.html

[12] *History - The Kamailio SIP Server Project* [online]. [cit. 2019-03-16]. Dostupné z: https://www.kamailio.org/w/history/

[13] *Kamailio SIP Server* [online]. [cit. 2019-03-16]. Dostupné z: https://www.kamailio.org/w/

[14] *Features - The Kamailio SIP Server Project* [online]. [cit. 2019-03-16]. Dostupné z: https://www.kamailio.org/w/features/

[15] *Kamailio SIP Server v3.2.0 Development Guide* [online]. [cit. 2019-03-17]. Dostupné z: http://www.asipto.com/pub/kamailio-devel-guide/

[16] *Kamailio SIP Server v5.1.x (stable): Core Cookbook* [online]. [cit. 2019-03-18]. Dostupné z: https://www.kamailio.org/wiki/cookbooks/5.1.x/core

[17] *RTPProxy* [online]. [cit. 2019-03-20]. Dostupné z: http://www.rtpproxy.org//about/

[18] *Rtpproxy(8): RTP Proxy Server - Linux man page* [online]. [cit. 2019-03-21]. Dostupné z: https://linux.die.net/man/8/rtpproxy

[19] *GitHub - sipwise/rtpengine: The Sipwise media proxy for Kamailio* [online]. [cit. 2019-03-21]. Dostupné z: https://github.com/sipwise/rtpengine

[20] *SER module mediaproxy - VoIP-Info* [online]. [cit. 2019-04-01]. Dostupné z: https://www.voip-info.org/ser-module-mediaproxy/

[21] *Network Topology Icons* [online]. [cit. 2019-04-03]. Dostupné z: https://www.cisco.com/c/en/us/about/brand-center/network-topology-icons.html

[22] *Install MariaDB 10.3 on Ubuntu 18.04 and CentOS 7 - Computingforgeeks* [online]. [cit. 2019-04-03]. Dostupné z: https://computingforgeeks.com/install-mariadb-10-on-ubuntu-18-04-and-centos-7/

[23] *Install Kamailio On Debian* [online]. [cit. 2019-04-04]. Dostupné z: http://kamailio.org/docs/tutorials/devel/kamailio-install-guide-deb/

[24] *Rtpproxy Module* [online]. [cit. 2019-04-05]. Dostupné z: https://kamailio.org/docs/modules/5.1.x/modules/rtpproxy.html

[25] *Download | MediaProxy* [online]. [cit. 2019-04-22]. Dostupné z: http://mediaproxy.ag-projects.com/download/

[26] *Welcome to SIPp* [online]. [cit. 2019-04-08]. Dostupné z: http://sipp.sourceforge.net/

[27] *SIPp* [online]. [cit. 2019-04-09]. Dostupné z: http://sipp.sourceforge.net/doc/reference.html

[28] *Installation - SIPp 3.5 documentation* [online]. [cit. 2019-04-20]. Dostupné z: https://sipp-wip.readthedocs.io/en/latest/installation.html

A Obsah přiloženého CD

Přiložené CD obsahuje následující soubory:

1. Konfigurační soubory:
    - kamailio.cfg
    - kamctlrc
    - rtpengine.conf
2. SIPp scénáře:
    - uac.xml
    - uas.xml
3. PCAP soubory:
    - g711a.pcap
    - g729.pcap